1

Research Product 89-19

Human Operator Simulator (HOS) IV User's Guide

June 1989

Manned Systems Group
Systems Research Laboratory

U.S. Army Research Institute for the Behavioral and Social Sciences

Approved for public release; distribution is unlimited.

U.S. ARMY RESEARCH INSTITUTE FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency Under the Jurisdiction of the Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON Technical Director	JON W. BLADES COL, IN Commanding

Research accomplished under contract
for the Department of the Army

Analytics, Inc.

Technical review by

Christine R. Hartel
Michael Young

NOTICES

FINAL DISPOSITION: This Research Product may be destroyed when it is no longer needed. Please do not return it to the U.S. Army Research Institute for the Behavioral and Social Sciences.

NOTE: This Research Product is not to be construed as an official Department of the Army document, unless so designated by other authorized documents.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
--

2a. SECURITY CLASSIFICATION AUTHORITY
--

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE
--

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
distribution is unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
--

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ARI Research Product 89-19

6a. NAME OF PERFORMING ORGANIZATION
Analytics, Inc.

6b. OFFICE SYMBOL *(If applicable)*
--

7a. NAME OF MONITORING ORGANIZATION
U.S. Army Research Institute

6c. ADDRESS *(City, State, and ZIP Code)*
2500 Maryland Way
Willow Grove, PA 19090

7b. ADDRESS *(City, State, and ZIP Code)*
5001 Eisenhower Avenue
Alexandria, VA 22333-5600

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
U.S. Army Research Institute for the Behavioral and Social Sciences

8b. OFFICE SYMBOL *(If applicable)*
PERI-SM

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F33615-~~8~~6-C-19

8c. ADDRESS *(City, State, and ZIP Code)*
5001 Eisenhower Avenue
Alexandria, VA 22333-5600

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO. (1204)	WORK UNIT ACCESSION NO.
62717A	790	121	H2

11. TITLE *(Include Security Classification)*
Human Operator Simulator (HOS) IV User's Guide

12. PERSONAL AUTHOR(S) Harris, Regina (Analytics, Inc.); Kaplan, Jonathan (ARI); Bare, Christopher; Iavecchia, Helene; Ross, Lorna; Scolaro, Dan; Wright, Douglas (Analytics, Inc.

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM 1987 TO 1988

14. DATE OF REPORT *(Year, Month, Day)*
1989, June

15. PAGE COUNT
267

16. SUPPLEMENTARY NOTATION The contract for performing the HOS IV project belonged to the U.S. Air Force Human Resources Laboratory, Wright-Patterson Air Force Base, Dayton, Ohio 45433. Michael Young and Christine Hartel are Contracting Officer's Representatives.

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Simulation Modeling
Human factors, Interface evaluation,
HOS Performance modeling. (SDW)

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This is the user's guide for the Human Operator Simulator (HOS) IV. HOS IV was developed to aid in the design and evaluation of interfaces between operators or maintainers and weapon system hardware and software. HOS IV can be used to simulate manned systems on IBM-AT or compatible computers by producing both system and human performance estimates based, in part, on micromodels of basic human processes. A set of micromodels is included in HOS. The user's guide provides the information required to install HOS, create new simulations, create new micromodels, edit existing simulations and micromodels, and run HOS simulations.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED □ SAME AS RPT. □ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
Jonathan D. Kaplan

22b. TELEPHONE *(Include Area Code)*
(703) 274-8943

22c. OFFICE SYMBOL
PERI-SM

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Research Prcduct 89-19

Human Operator Simulator (HOS) IV User's Guide

Regina Harris
Analytics, Inc.

Jonathan D. Kaplan
U.S. Army Research Institute

Christopher Bare, Helene Iavecchia, Lorna Ross, Dan Scolaro, and Douglas Wright
Analytics, Inc.

Manned Systems Group
John F. Hayes, Chief

Systems Research Laboratory
Robin L. Keesee, Director

U.S. Army Research Institute for the Behavioral and Social Sciences
5001 Eisenhower Avenue, Alexandria, Virginia 22333-5600

Office, Deputy Chief of Staff for Personnel
Department of the Army

June 1989

Army Project Number
2Q162717A790

Human Performance
Effectiveness and Simulation

FOREWORD

The U.S. Army Research Institute for the Behavioral and Social Sciences (ARI) and the U.S. Air Force Human Resources Laboratory have developed a simulation technique for evaluating manned systems based on design, performance of operators, and activities of the environment. This method is called the Human Operator Simulator (HOS) IV. It is a substantial alteration of the original version of HOS developed by the U.S. Navy. HOS-IV runs on an IBM-AT or equivalent personal computer and can be used at any stage of system development to model, and thus evaluate, manned developmental or non-developmental items. HOS IV is one of a number of programs ARI and AFHRL are using to advance the state of the art of manned system design and evaluation. This Guide provides the information needed to build, alter, and analyze HOS models.

Edgar M Johnson

EDGAR M. JOHNSON
Technical Director

CONTENTS

1. OVERVIEW

WHY HOS?

The rapid and continuing evolution of complex technology has been accompanied by a host of problems involving the relationship between humans and systems. Designing a system involves complex decisions that try to balance a multitude of constraints and criteria. A system designer is generally supplied with a set of mission goals, major system components (sensors, processors), the expected environment (e.g., likely enemy forces), an initial operator/system function allocation, and a set of operator and system procedures. Using this information, the system designer must determine the ability of the operator/system to accomplish mission goals and to establish if a particular design has the correct balance of system and operator responsibilities. Ideally, this process would occur in the early states of system design where it is possible to affect the design in a cost-effective manner.

A need existed for a tool to assist in the design process to determine whether the human operator can operate the system as intended. The goal was to construct a tool that would permit system developers to quickly evaluate the performance of a system and access the ability of the human to effectively use the system early in the design process. Therefore, the Human Operator Simulator (HOS) approach was developed to be a generic, analytical approach to simulate human-machine systems. HOS utilizes a performance decomposition process to simulate the cognitive, perceptual, and motor activities of a human operator while the operator is manipulating a specific system within a defined environment.

HOS-IV

HOS was originally conceived in 1970 and has undergone a series of major upgrades resulting in the latest version, HOS-IV. HOS-IV has been specifically designed to operate in a microcomputer environment and contains numerous improvements based upon experience with previous versions of HOS. HOS has been developed to be a general purpose simulation tool for modeling human operators, the systems that they operate, and the environment in which the system and operator are working. The required inputs to the model are descriptions of the system design, procedures for using thc system, human operator characteristics, and a mission scenario. A set of operator micromodels are available to the HOS user to assist in the development of the simulation. These micromodels contain algorithms, based on experimental literature, which can predict the timing and accuracy of basic human cognitive, perceptual, and psychomotor actions.

HOS SIMULATIONS

HOS provides a capability for a comprehensive human-system simulation. Figure 1 presents the components developed for an application simulation. During a typical implementation, such as a simulation of a radar operator and crewstation, the HOS analyst first determines the allocation of functions between the human operator and the machine. The analyst then describes:

- The environment (e.g., number, location, speed, and bearing of enemy targets);
- The hardware system (e.g., radar sensor and signal processors, displays, and controls); and
- The operator procedures and tactics for interacting with the system and for accomplishing mission goals.

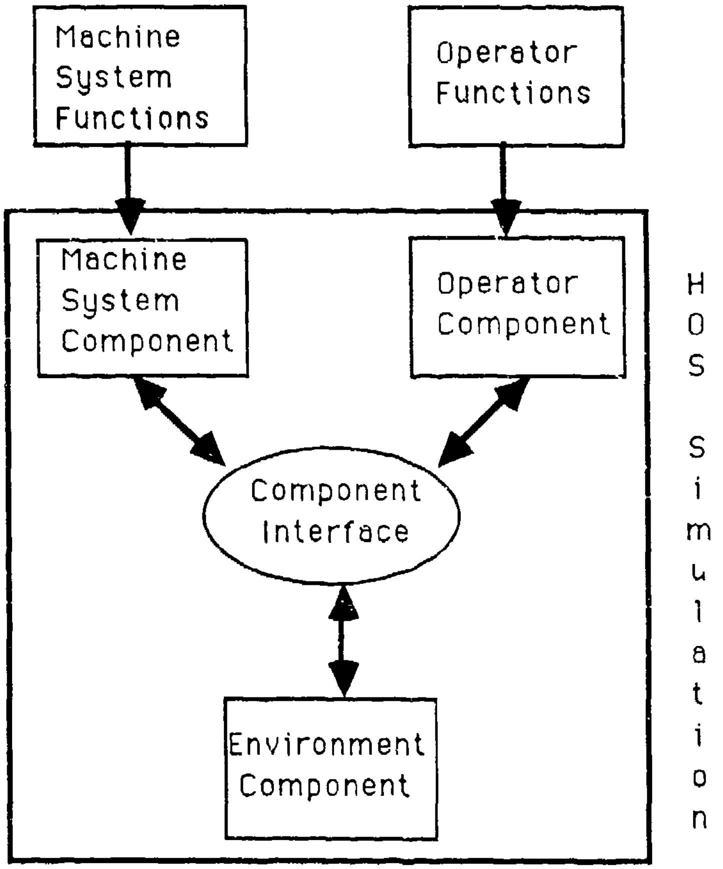


Figure 1. HOS Simulation Components

In addition, the definitions of the interactions among these three components — the operator, system, and environment — are important in HOS. For example, in a radar system simulation, a routine representing the hardware and environment interactions would determine which enemy targets were within the radar detection range at any given temporal snapshot. A routine representing the interactions between the operator and environment could determine the effects of heat, cold, drugs, or other stresses on human performance timing and accuracy. The operator-system models could establish the time and accuracy of an operator performing such tasks as reading alphanumeric information from displays, manipulating controls, searching for targets in a particular field-of-view, or physically moving objects from one location to another.

HOS RESULTS

The execution of HOS-IV results in a sequence of operator decisions about what to do at each point in time based on moment-to-moment mission events and predefined tactics and procedures. Once the simulated operator decides what procedure to do next, HOS micromodels are invoked to carry out specific actions such as reading information from a display or manipulating a control. The HOS micromodels are based on data derived from the human performance literature and are available to the user through micromodel libraries. The result of a simulation is a detailed timeline of operator, hardware, and environmental events which can be summarized and analyzed to support a broad variety of purposes. HOS-IV has been designed to be a production tool to support system designers as well as human factors engineers to evaluate human-system performance.

HOS FEATURES

An important feature of HOS-IV is that it utilizes rule-based concepts. Inputs to HOS-IV include a set of rules which activates a set of actions when conditions are appropriate. Defining these rules facilitates a "top-down" approach to simulation since it allows the user to design the simulation flow independent of the implementation of low level actions and models. It also allows the simulation to more closely mimic reality since operators usually make decisions based on an implicit or explicit set of rules when responding to a particular situation.

ABOUT THIS MANUAL

This document presents information about the capabilities and features of HOS-IV. The remainder of this manual is divided into the following sections:

2. **Simulation Basics**

   Provides an overview of the components required to develop a HOS simulation

3. **HOS User Interface**

   Describes the components of the HOS user-computer interface and how to use them.

4. **Getting Started**

   Describes the hardware and software environment required for HOS and the procedure for installing HOS on your system. It also explains how to start the HOS software.

5. **Defining a Simulation**

   Describes how to use the HOS-IV system to define a simulation and describe basic simulation parameters.

6. **Using the HOS Editors**

   Describes how to use the HOS editors to define and modify objects, events, rules, and actions.

7. **HOS Action Language**

   Describes the components of the HOS Action Language (HAL).

8. **Operator Models**

   Describes the operator models included in the HOS-IV system and how to integrate them into your simulation.

9. Creating and Running a Simulation

Describes how to build and run a HOS simulation.

10. Simulation Results

Describes the simulation reports and how to obtain and interpret these reports.

11. Simulation Examples

Presents examples of HOS simulations and describes how they were constructed and analyzed.

Appendix A. General Installation

Provides general instructions on installing HOS on non-standard computers.

Appendix B. HOS Micromodels: Default Time and Error Parameters

Lists the default parameter values of HOS micromodels that are used in predicting human performance times and errors.

Appendix C. Troubleshooting

Lists typical errors encountered during HOS-IV system use as well as guidance for correcting error conditions.

2. SIMULATION BASICS

HOS-IV is a highly flexible simulation facility which allows the user to develop a simulation incrementally. The operator characteristics, hardware components, and environmental factors can be developed and tested independently. Furthermore, each component can be simulated at varying levels of detail. For example, the operator characteristics can be simulated simply by charging a fixed time for the operator to respond to a system alert. Or more complicated perceptual and fatigue models such as those available from the set of HOS micromodels can be accessed to determine operator alert processing times and errors.

The user can **build** a simulation layer by layer, developing components to any level of detail desired. Thus, the user can define any or all of the following components in any order:

- The external world, or environment (enemy aircraft, emitters);
- The system (sensors, processors);
- The operator system interface (displays, controls, procedures); and
- The operator activities (how the operator makes decisions, how the operator accesses necessary information, how the operator executes decisions, etc).

The user also defines events which can affect any of these layers during the simulation. Furthermore, a system failure can be specified to occur at some time during the simulation.

HOS SIMULATION STRUCTURE

To further define the characteristics of the environment, hardware, and operator, the following information is specified:

- Objects,
- Events,
- Rules, and
- Actions.

Objects

The knowledge about the important entities in the simulation and their distinctive features are simulated using an object-characteristic structure. Each entity to be modeled in the simulation (e.g., displays, controls, sensors, aircraft, etc.) is described as an object. Each object has an associated list of characteristics such as size, color, status. This object-characteristic structure provides the user with the capability to define what is important to the simulation at any level of detail.

Characteristics

The list of characteristics associated with each object describes the features of the object relevant to the simulation. Each characteristic is assigned a value which indicates its state at a particular point in the simulation. For example, the **status** characteristic of the **generator** object can have a value **on** or **off**. Furthermore, objects with the same set of characteristics can be distinguished from each other by the values of their characteristics. For example the characteristic **color** differentiates a **dial** object which is **red** from a **dial** which is **blue**.

Object Editor

The Object Editor is used to modify objects and the characteristics, as well as the values for those characteristics. For example, the analyst might define an object **switch**, with the characteristic **status**, which has the initial value **off**. Later, during the simulation, the status of the switch might be changed to **on**.

Level of Detail

Objects can be defined at various levels of detail. For example, the object **emergency indicator light** can have one characteristic, i.e., **status**, with a value of on or off. Another object may have several characteristics, such as enemy emitter with characteristics for location, speed, bearing, frequency, pulse width, pulse repetition frequency, and pulse mode, along with assigned values. The values of the characteristics indirectly control the flow of events in the simulation by providing information to the rules and actions.

Object Example

The definition of an object to represent a radar is illustrated below:

Object Name: Radar

Characteristics	Initial Value
Status	off
Mode	automatic
X-location	46.78
Y-location	124.54

The important features of the radar are defined as characteristics and include status, mode of operation, and x and y location in the workspace. The initial values contain the state of the characteristic at the beginning of the simulation. For example, when the simulation starts, the status of the radar would be off and the mode automatic.

Object Sets

A group of objects which are of the same type (i.e., they have the same **characteristics**) can also be simulated. For example, a group of emitters share the same set of characteristics (location, name, frequency, etc.). If the analyst wished to simulate a group of 50 emitters, it would be very time-consuming to repeatedly enter location, name, etc. for each emitter. To alleviate this problem, HOS-IV allows the definition of groups of objects, called **sets.** Each object in the set has exactly the same characteristics. However, the value of the characteristics can vary among members of the set.

EVENTS

Events represent external occurrences that affect the simulation. They simulate processes that are not originated by the operator but which have an impact on the course of the simulation. Events are defined by the event name, the time of the event, and the name of the action that is to be triggered at the event time. Typical events are special tasking messages communicated to the operator from a higher command, hardware system failures, or environmental changes (e.g., weather). For example:

At Time:	00:05:00.000
Event Name:	Electrical storm causes power outage.
Do:	system_failure *(action name)*

This event simulates a system failure at 5 minutes representing a power outage caused by an electric storm.

RULES

Rules define what actions are to be taken depending on specified conditions. Rules are defined by a starting and ending condition, the name of the action to be invoked if the starting condition is true, and a priority assignment. Rules are separated into one of three categories (or queues) — operator, hardware, and environment. Rules can be grouped according to precedence or mission phase by the assignment of a priority. Hardware and environment can contain a maximum of 100 rules each. Rule 99 has the highest priority; 0 has the lowest. 1000 operator rules can be defined and the operator rules are pooled into 10 groups (0 - 9). Each operator group can contain 100 rules. Operator rule 999 has the highest priority; operator rule 001 has the lowest.

Each rule is defined by three clauses:

- An IF clause that specifies a rule initiation criteria,
- A DO clause that specifies the action to be taken if the initiation criteria is true, and
- An UNTIL clause that specifies the rule completion criteria.

Rule Example

An example of a rule is:

IF color OF alert_light EQUALS yellow

DO process_yellow_alert

UNTIL color OF alert_light NOT_EQUAL_TO yellow

This rule will be invoked whenever the color of the alert light is yellow. When the rule is invoked, the action process_yellow_alert will be performed.

Rule Processing

If the starting conditional statement is true, the action named in the DO clause will be executed is the rule is active. The action will be executed at each simulation time interval until the UNTIL condition is true or the rule is suspended. Rules, or groups of rules, can be declared active or inactive by actions at any point during the simulation. At each simulation time unit, the condition statements of all active rules will be evaluated to determine what actions will be executed or terminated. The rules will be processed in the sequence specified by the priority assignment. This precedence relationship is particularly useful as actions invoked by a high priority rule may suspend lower priority rules. This capability permits rules to be categorized by mission critical tasks as well as mission phase. All active rules represent the processes that are occurring in parallel during a simulation time interval.

ACTIONS

Actions describe the steps required to accomplish a process by the operator, system, or environment. An action can be invoked in one of the following ways:

1. By a DO clause in a rule,
2. By another action, or
3. By an event.

Actions update the values of object characteristics, invoke other actions, and activate or suspend rules. The values of the characteristics of objects can only be changed by an action.

Actions are defined using a small set of standard verbs (e.g., DO, SET, SUSPEND) known as the HOS Action Language (HAL). Verbs are the 'primitive' building blocks with which the user constructs complex and application-specific actions. The HOS verbs are summarized below with more detail provided in Section 7.

Informational Verbs

Informational verbs are used to provide information about the simulation both within the action and to external sources such as the printer and simulation files. Informational verbs include:

- COMMENT — used to insert text comments at any point in an action,
- DEFINITIONS — used to categorize each variable used within an action,
- FILE — used to store values in a file for later printing or processing, and
- PRINT — used to immediately print simulation values.

Object Manipulation Verbs

Object verbs are used to retrieve and manipulate the values of the characteristics of objects. They are also used to locate and access members of object sets. Object manipulation verbs include the following:

- GET — used to access the current value of a characteristic of an object,
- PUT — used to store a value in a characteristic of an object, and
- RETRIEVE — used to access specific members of an object set.

Computational Verbs

Computation verbs are used to specify calculations The computation verb, SET, is described below:

- SET — used to change the value of a variable and perform calculations. It provides access to standard mathematical, logarithmic, trigonometric functions, and random numbers.

Rule Manipulation Verbs

Rule manipulation verbs are used to add or suspend rules from the active rule lists. Rule manipulation verbs include the following:

- START — used to add rules to the active rule lists, either a single rule, group of rules, or all rules and
- SUSPEND — used to suspend rules from simulation processing, either a single rule, group of rules, or all rules.

Action Manipulating Verbs

Action manipulating verbs are used to call other actions and include the following:

- USING...DO — is used to call other actions and can optionally pass parameters to the invoked action and

- RECEIVE — is used to name and receive parameters passed from other actions.

Conditional Verbs

Conditional verbs are used for controlling the execution of a series of statements based upon the truth of a condition. Two conditional verbs are provided:

- IF...THEN — used to controll the execution of a series of statements based upon whether the expressed condition is true or not and
- WHILE...THEN — used to control the execution of a series of statements as long as the condition is true.

Terminating Verbs

Terminating verbs are used to end the simulation (END_SIM) and end (END) actions.

Special Verbs

HOS-IV contains provision for including models written using the C language (Microsoft C Version 4.0). These models can be directly integrated into the HOS actions. The START_C_CODE and END_C_CODE verbs are used to enclose the C code.

MICRO-MODELS

HOS includes operator micromodels which generate performance time and an indication of success/failure when appropriate. The current version of HOS-IV contains cognitive (recall, mental computations), perceptual (visual and auditory), and psychomotor (anatomy movement) models which are based on experimental data from the human performance literature. The HOS analyst is able to tailor the models to the needs of a particular application by modifying key parameters of the micromodel or incorporating a new model. All models included in HOS-IV are written utilizing the same HAL language as used to define simulation actions. The models are described in detail in Section 8.

HOS OUTPUTS

The HOS-IV outputs include:

- A timeline of events for the operator, system, and environment,
- User-defined measures of effectiveness, and
- Standard analysis, such as:
  - Mean time to complete an action,
  - Number of times an action is performed,
  - Proportion of the operator's time spent on each action.

HOS-IV allows for user-controlled detail of simulation outputs. When building the object library, the user can identify objects whose values are to be tracked throughout the simulation. User-defined measures of effectiveness, such as the number of contacts processed per hour or a history of error rates, can also be defined and stored in a separate simulation file at simulation end. With this data, the user can identify operator and system bottlenecks and determine periods of operator overload and the circumstances surrounding those overload periods.

3. HOS USER INTERFACE

This section describes how the user interacts with the HOS system.

CURSORS

HOS uses two distinct cursors to represent the focus of attention for the user and to point to a precise point on the screen:

- The **mouse cursor** is controlled by the mouse and always represents the last mouse screen location. When the user moves the mouse, the mouse cursor moves proportionately.
- The **keyboard cursor** represents the location of any keyboard actions.

Mouse Cursor

The mouse cursor is a pointing device used to select commands from menus, to control cursor movement, and to manage file scrolling. It is represented on the screen as an arrow (↗). The user moves the mouse cursor to the desired location by moving the mouse in the desired direction. Every mouse movement moves the mouse cursor in exactly the same way. The mouse actions are described further below in the section labeled **Mouse.**

Text Cursor

The text cursor indicates where next keyboard stroke will be entered and is represented on the screen as a rectangle the size of a single character (▮). The user can use the arrow keys or the mouse to indicate where the text cursor should be positioned. Each subsequent keyboard stroke will be inserted at the location of the text cursor.

INPUT DEVICES

The HOS input devices consist of two complementary devices — a keyboard and a mouse. The keyboard is used mainly for entering text and numbers, while the mouse is used for specifying menu options, controlling the cursor, selecting information, and specifying insertion points.

Keyboard

The keyboard is used to enter alphanumeric data and as an alternative to the use of the mouse to move the text cursor between input fields on dialog boxes.

Numeric and Cursor Movement Keypad

The numeric keypad includes keys for the numbers zero through nine arranged in an adding machine format; it also has keys for special functions (such as a minus sign, equal sign, etc.) and is used to speed entry of numeric information. Superimposed on the numeric keypad is the cursor movement keypad. The cursor movement keypad contains an up-arrow, down-arrow, right-arrow, left-arrow, home, and end keys and are used to control cursor movement within a window as an alternative to the use of the mouse. The user controls the functioning of the keypad as either a numeric pad or cursor movement pad through the use of the **NUM LOCK** key. Above the num lock key is a small red dot light. If the light is lit, then the keypad is functioning as a numeric pad; otherwise it functions as a cursor movement pad. The special function keys are used to select menu options as an alternative to the use of the mouse for experienced users with certain HOS modules.

Special Keys

The following keys have special functions as described within the HOS modules indicated in parentheses:

Enter (↵)

Enter (↵) is used for the following:

1. Move the text cursor and any subsequent text to the next line (Action and Object Editor modules) and
2. Move the text cursor to the next data entry field in the dialog window (All other modules).

Backspace (←)

Backspace (←) deletes the character to the left of the text cursor. If the text cursor is positioned at the top, leftmost character in a window, subsequent depressions of the backspace key are ignored (All modules).

Tab (|←)

Tab (|←) is used for the following:

1. Insert up to five blank characters in the text depending upon the current cursor position. If the entry of the blank characters causes the width of the line to exceed the screen width, the text cursor and any subsequent text will be moved to the next line (Action Editor module) and
2. Move the text cursor to the next input field in the dialog window (All other editors).

~ and '

The key containing the ~ (upper case) and ' (lower case) is used to generate the underscore character __ regardless of the status of the shift and/or shift lock keys.

Special numeric keypad/cursor movement keys

The following keys on the cursor movement pad have special functions as described for the indicated HOS modules:

HOME

HOME (above 7 key on numeric pad) is used for the following:

1. Move the text cursor to the leftmost character in the first entry field in the dialog window (Object Editor),
2. Move the text cursor to first screen containing text (Action Editor) and maintain the relative position of the text cursor, and
3. Move the text cursor to the leftmost character in a text entry box (All other modules).

Up-arrow (↑)

Up-arrow (↑) — (above 8 key on numeric pad) moves the text cursor up one line. If the current line is the top line on the page, the depression of the up arrow scrolls the page. If the current line is the first line then subsequent depressions of the up-arrow are ignored (Action Editor module).

PgUP

PgUP (above 9 key on numeric pad) moves the text cursor to the previous page of text and maintains the relative position of the text cursor on the page. The top line of the previous page becomes the bottom line of the current page (Action Editor module).

Right-arrow (→)

Right-arrow (→) (above 6 key on numeric pad) moves the text cursor one character to the right (All modules).

Left-arrow (←)

Left-arrow (←) (above 4 key on numeric pad) moves the text cursor one character to the left (All modules).

END

END (above 1 key on numeric pad) is used for the following:

1. Move the text cursor to the rightmost character in the last entry field in the dialog window (Object Editor module),
2. Move the text cursor to last screen containing text and maintain the relative position of the cursor (Action Editor module), and
3. Move the text cursor to the right most character in a text entry box (All other modules).

PgDN

PgDN (above 3 key on numeric pad) moves the text cursor to the next page of text and maintains the relative position of the text cursor on the page. The bottom line of the previous page becomes the top line of the current page (Action Editor).

Down-arrow (↓)

Down-arrow (↓) (above 2 key on numeric pad) moves the text cursor down one line. If the current line is the last line of the page, the down-arrow will scroll the text down one line. If the current line is the last text line, subsequent depressions of the down-arrow will be ignored (Action Editor).

Functions Keys

The following functions keys are used for indicated processes within the Action Editor:

- F1 — Begin text marking for cut/copy operation.
- F2 — End text marking for cut/copy operation.
- F3 — Cut text.
- F4 — Copy text.
- F5 — Paste text.
- F6 — Clear text.

MOUSE

The mouse is used as a pointing device to select commands from menus, to control cursor (pointer) movement and to manage file scrolling. In HOS-IV, the standard pointer is an arrow (↗). Every move of the mouse moves the pointer in exactly the same way. The following terms describe various actions associated with using the mouse. The HOS software was written to treat all of the mouse buttons identically so it does not matter which of the buttons you push.

The following terms describe various actions associated with using the mouse.

Mouse Clicking

Clicking is used to indicate that the mouse pointer is located where you want it to be and to indicate selections. To click an item:

1. Use the mouse to move the mouse pointer so that the tip of the pointer is on the desired item.
2. Press and immediately release the mouse button without moving the mouse.

Mouse Pressing

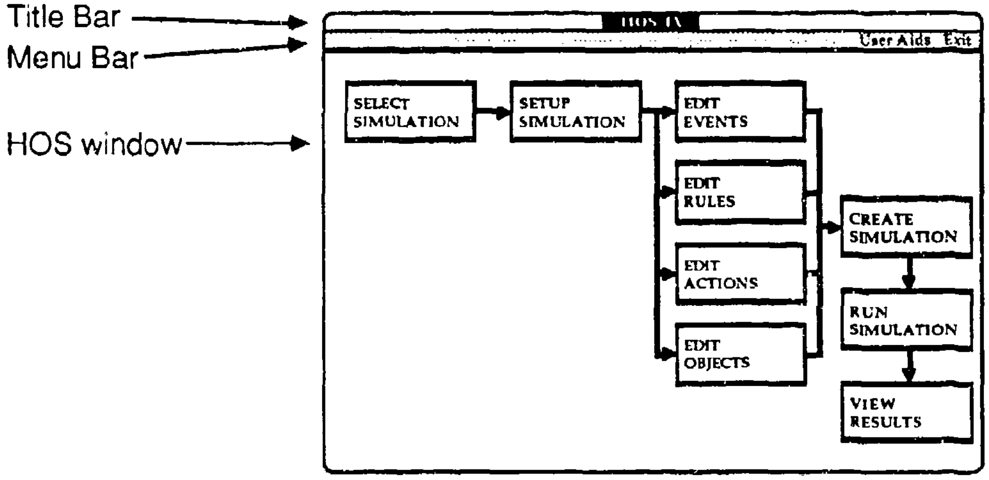
Pressing the mouse involves positioning the pointer by holding down the mouse button without moving the mouse.

Mouse Dragging

Dragging is used to extend a selection by positioning the pointer with the mouse, holding down the mouse button, moving the mouse to a new position, then releasing the button.

MAIN HOS SCREEN

The main HOS screen is illustrated below:



The screen is divided into three major areas:

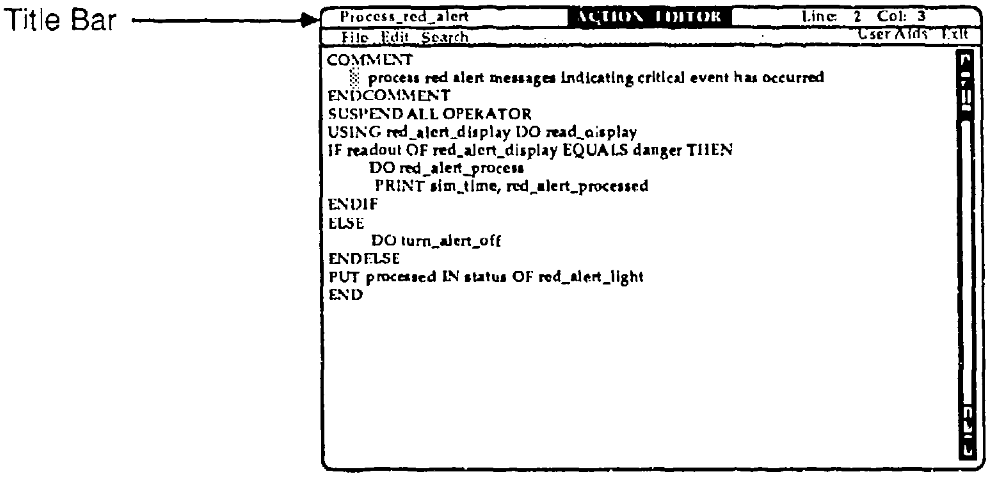
1. The **title bar** is the top line of the screen. The title bar contains information about the current HOS activity including name and status information. The current HOS activity for the screen shown above is HOS-IV.
2. The **menu bar** located on the second line of the screen contains a list of options for the current HOS activity. The menu bar on the screen above contains two options — **User Aids** and **Exit.**
3. The **HOS window** occupies the remainder of the screen. The contents of the HOS window vary depending upon the current HOS activity. The screen above illustrates a set of pushbuttons that shows the user the progression of HOS activities that must be performed for a HOS simulation and allows the user to select which activity to perform next through the use of pushbuttons.

The Title Bar

The title bar is continuously displayed at the top of the screen with black letters on a white background. The title bar presents information about the current HOS function to the user. The user does not interact with the title bar.

HOS Title Bar Scieen

The title bar is illustrated as below:



```
COMMENT
    process red alert messages indicating critical event has occurred
ENDCOMMENT
SUSPEND ALL OPERATOR
USING red_alert_display DO read_oisplay
IF readout OF red_alert_display EQUALS danger THEN
      DO red_alert_process
       PRINT sim_time, red_alert_processed
ENDIF
ELSE
      DO turn_alert_off
ENDELSE
PUT processed IN status OF red_alert_light
END
```

The title bar contains three pieces of information:

1. The name of the current activity displayed on the left side of the top line, e.g., the name of the action currently being edited. For the screen shown above, the name of the action is **Process_red_alert**.
2. The name of the current HOS function is centered. For the screen shown above, the current HOS function is the **Action Editor**.
3. Information on the current status of an activity on the rig'it side of line with initial caps, if required e.g., the line count and column position for the file being processed by the action editor). For the screen shown above, the status information indicates that the current line being edited is **line 2** and the column is **3**.

The Menu Bar

The second line of the display contains the menu bar which lists the titles of options that are available to the user. From the menu bar, the user can select from any of the currently available options. The options displayed on the menu bar will vary between HOS modules but always contain **User Aids** and **Exit** as the last two options.

The HOS Window

The HOS window is used to conduct a dialog with the user. It either displays information to the user or displays an input form for the user to supply the information HOS requires. The contents vary dependent upon the current function. Within the HOS window, a variety of components have been developed for the user to specify particular simulation data items or supplying additional information required before a system command can be processed. The HOS windows are categorized as follows:

- Message Windows,
- Dialog Windows,
- Information Windows, and
- Text Entry Windows.

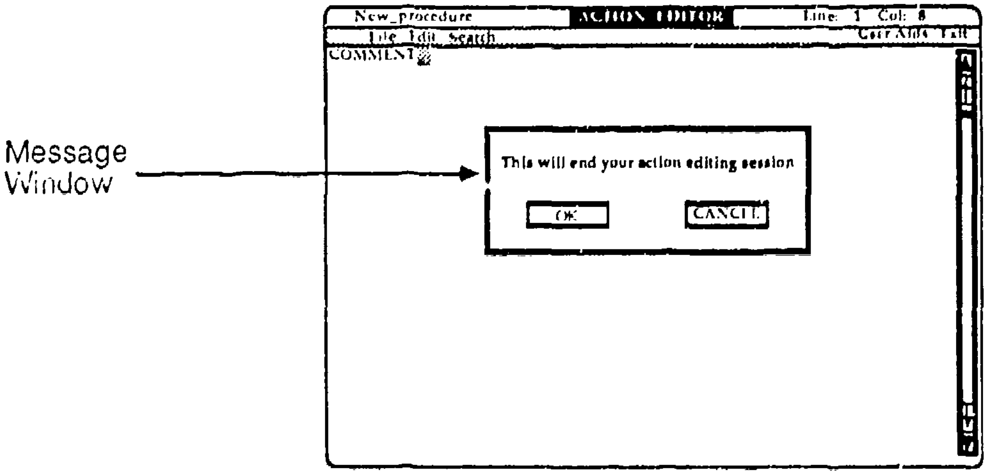
Each of these windows is described in more detail in the following sections.

Message Windows

A message window, as illustrated below, presents informative messages about the current system action. The user can indicate whether subsequent actions should occur or be cancelled. It is displayed on top of the contents of the previous screen and is enclosed in a double line rectangle. The options available to the user are displayed as pushbuttons and one of the pushbuttons always contains a CANCEL option that permits the user to cancel the current request and resume the previous activity. The mouse is used to move the mouse cursor to select the pushbutton containing the desired action.

HOS Message Window

The message window illustrated below shows that the user has selected the command to terminate the Action Editor. It will be displayed for the user to confirm the termination. If the user wishes to terminate the action editor, the **OK** pushbutton will be selected; otherwise the **CANCEL** pushbutton will be selected. If cancel is selected, the termination request is ignored and the Action Editor continues processing user commands.



This will end your action editing session

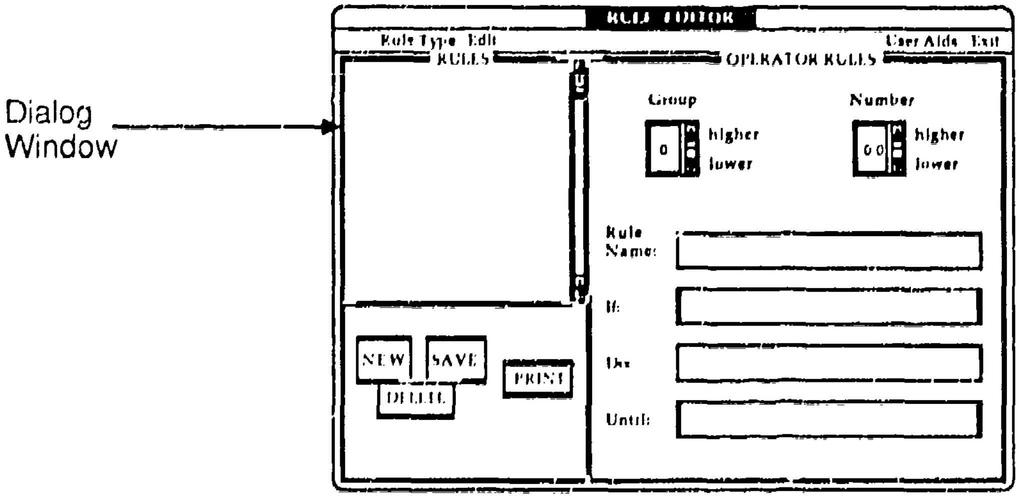
Dialog Windows

Dialog windows are used for the specification of a group of related information. They are mainly used for specifying particular simulation data items or supplying additional information required before a system command can be processed.

Dialog windows, as shown below, allow the user to enter necessary information in pre-defined fields consisting of pushbuttons, click boxes, and text entry boxes. The title of the dialog window is displayed in upper case centered in the top line of the window. The text cursor is initially placed in the beginning of the first text entry box for the user to enter the indicated information. When the entry is completed, the user can depress the return key to move the text cursor to the next item in the sequence or, alternatively, use the mouse to move the mouse pointer to the desired field and click to obtain the text cursor in the desired location. In addition, the tab key can be used to move forward to the next text entry field. The various components of the dialog window are described in subsequent sections.

HOS Dialog Window

The dialog window shown below contains all the dialog components for the user to define a rule.

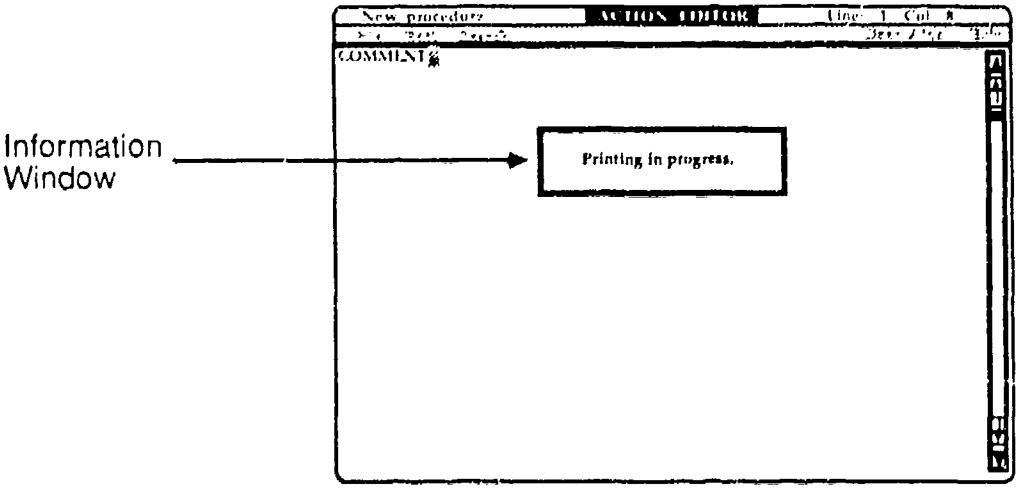


Information Windows

An information window presents messages about current system activity and is illustrated below. An information window differs from a message window in that the user cannot make any responses.

HOS Information Window

The information window shown below indicates the the user has requested a print command and that the printing is in progress. The window will remain on the screen until printing is completed.



COMMENT

Printing in progress.

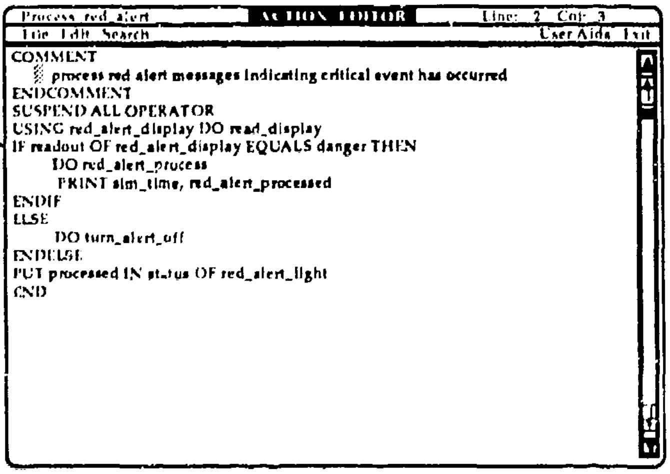
Text Entry Screens

A text entry screen is a user scrollable window in which the user can enter textual/numerical information in a free-format. A scroll bar is displayed on the right side of the window and is illustrated below. The rectangular text cursor is initially placed at the upper left corner of the window. The user can use the mouse or arrow keys (right, left, up, down, home, page up, and page down) to position the text cursor to the location where the next keyboard entry is to be placed. All key strokes are inserted at the current location of the text cursor. If the entry causes the length of the current line to exceed the display width, all text following the previous delimiter (space) is moved to the next line. The depression of a carriage return moves the text cursor and any text after it to the next line. The home key moves the text cursor to the first window of text; the end key moves the text cursor to the last window containing text.

HOS Text Entry Window

The text entry screen illustrated below shows the area beneath the title bar and menu bar that is available for the user to enter text information.

Text Entry Window



```
COMMENT
  process red alert messages indicating critical event has occurred
ENDCOMMENT
SUSPEND ALL OPERATOR
USING red_alert_display DO read_display
IF readout OF red_alert_display EQUALS danger THEN
      DO red_alert_process
       PRINT sim_time, red_alert_processed
ENDIF
ELSE
      DO turn_alert_off
ENDELSE
PUT processed IN status OF red_alert_light
END
```

DIALOG COMPONENTS

The HOS dialog consists of the following components:

- **Menu Bar** — a list of available options displayed on the second line of the display.
- **Pull-down Menu** — a submenu which is shown in a separate window displayed beneath the menu bar containing the list of commands available for a particular selection from the menu bar.
- **Pushbutton** — a boxed area of the screen that is used to invoke a function.
- **Scroll Bar** — a rectangular area of the screen that contains up and down arrows that are used to modify the current contents of the displayed window.
- **Dialog Box** — a rectangular area in which the user enters information associated with the displayed label.
- **List Box** — a rectangular area containing all the currently defined and available items in a separate window.

The components of the HOS dialog are described individually in subsequent sections.

Menu Bar

The **menu bar**, located on the second line of the screen, contains a list of the options available for the current HOS function as shown below. It is continuously displayed on the screen and contains white letters on a blue background. Submenus "pop-down" temporarily when the user selects an option from the menu bar, and the submenu may obscure the contents of the screen displayed below.

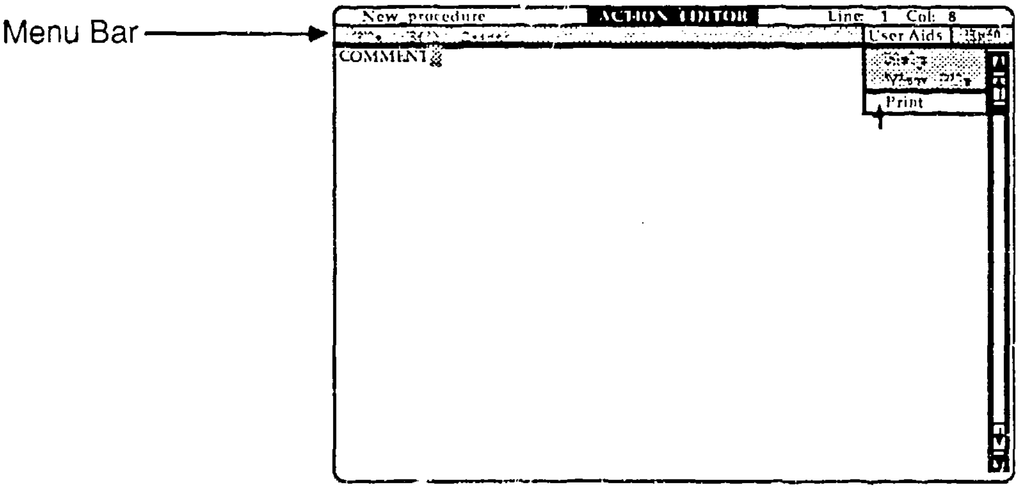
The submenus are displayed in a separate box beneath the menu bar showing all available options. To select an option from the menu bar, use the mouse to position the pointer anywhere on the desired title in the menu bar. Without moving the mouse, press and hold the mouse button (clicking).

Once the mouse button is pressed, the selected menu title will be shown in reverse video (white background and blue foreground) and a box containing the available commands will appear immediately beneath the menu title in a separate window. This box is referred to as a **pull-down menu** and is described below. The pull-down menu will disappear as soon as the mouse button is released.

In order to view all the pull-down menus, the user can drag the mouse across the menu bar and as each menu title is selected, the accompanying pull-down menu will be displayed. The contcnts of the menu bar will vary with the associated function but will always contain **User Aids** and **Exit** as the last two items.

Menu Bar Illustration

The illustration below contains an example of the menu bar that contains the following commands: File, Edit, Search, User Aids, and Exit.



Pull-Down Menus

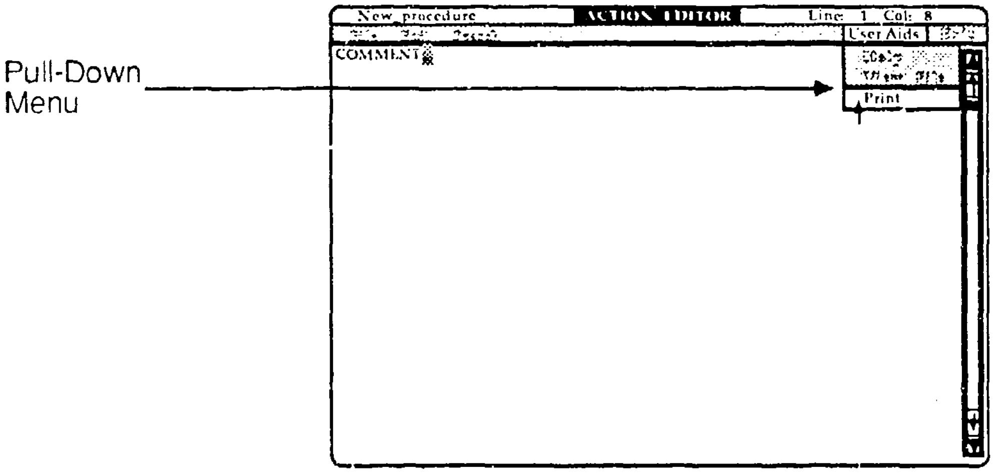
Pull-down menus are displayed in a box drawn in a separate screen window positioned directly beneath the title of the selected menu bar option as shown on a later page. The pull-down menu contains a list of the names of the available commands associated with the selected menu bar option.

To choose one of the listed commands in the pull-down menu, the mouse is used to move the mouse pointer to the displayed menu title on the menu bar. While the mouse button is held down, the mouse is used to move the mouse pointer to the desired command (dragging). When the mouse pointer is located over the selected command, the mouse button is released. As the mouse pointer moves to each command line, the currently selected command is highlighted in reverse video (white foreground and blue background). The command that is highlighted when the mouse button is released is invoked and the pull-down menu disappears.

If the mouse cursor is relocated within the menu bar line and the mouse button released, no action will occur and the pull-down menu will disappear. Similarly, if the mouse cursor is dragged outside of the pull-down menu window and released, the pull-down menu will disappear and no command will be chosen. The selected command may present additional menus or input templates or perform the selected task if all necessary information is available. If you decide not to select any command at this time, move the pointer outside of the pull-down menu area and release the mouse button.

Pull-Down Menu Illustration

The screen below illustrates the pull-down menu commands that are available when the user selects the **User Aids** option from the menu bar. The commands include: Help, View File, and Print. Print is the currently selected command since it is highlighted.

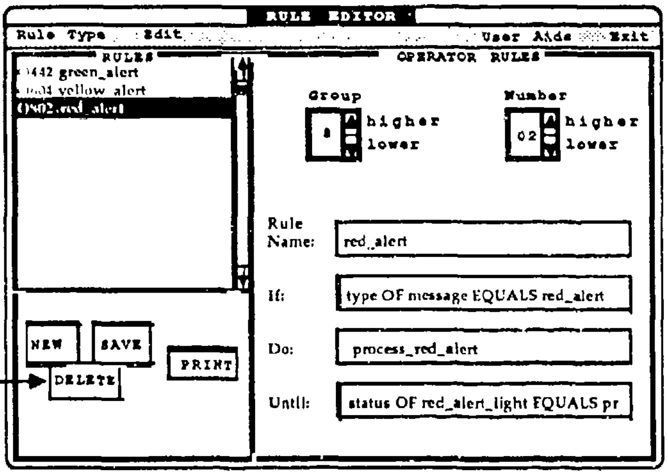


Pushbuttons

Pushbuttons are rectangular boxes in the HOS window portion of the screen that perform instantaneous actions as described by the text label. Pushbuttons are rectangles with double lines on the top and bottom of the box and single line on left and right as illustrated below. Each pushbutton contains a text label indicating its function. To use a pushbutton, move the mouse so that the mouse cursor is located anywhere within the rectangular area and click. The selected pushbutton area will be highlighted and the function immediately invoked.

Pushbutton Illustration

The screen below illustrates the rule editor dialog window that contains four pushbuttons: New, Save, Delete, and Print.



0442 green_alert
0?04 yellow_alert
0802 red_alert

Rule Name: red_alert

If: type OF message EQUALS red_alert

Do: process_red_alert

Until: status OF red_alert_light FQUALS pr

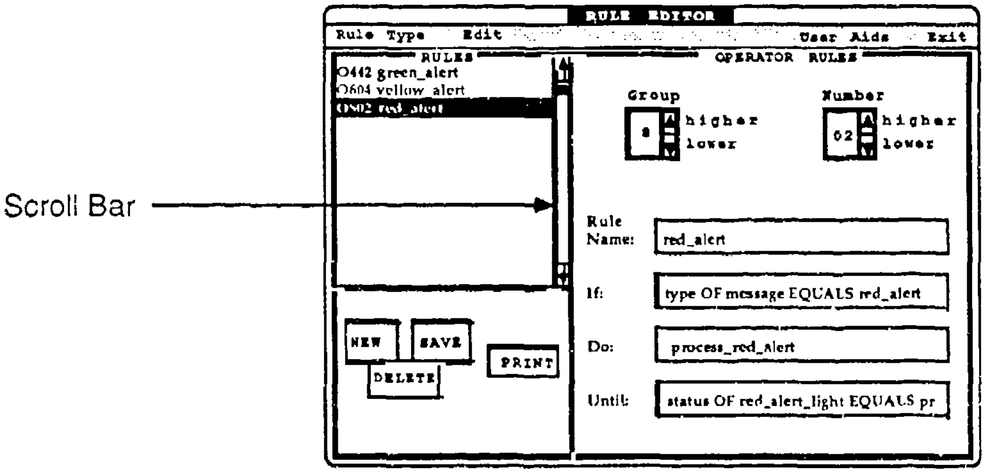
Pushbutton

Scroll Bar

Scroll bars are used to change that part of a list of items which is shown in the window. They are shown as rectangular boxes on the right side of the area to which they apply. Double red scroll arrows are used at the top and bottom of the scroll bar rectangle to indicate the direction the viewing area is to be moved. The top arrow (▲) is used to scroll up one line at a time; the down arrow (▼) is used to scroll down one line at a time. The second up arrow (↑) is used to scroll up one page at a time; likewise the top down arrow (↓) is used to scroll down one page at a time.

Scroll Bar Illustration

The scroll bar is illustrated below.



O442 green_alert
O604 yellow_alert
O802 red_alert

Group: 8

Number: 02

Rule Name: red_alert

If: type OF message EQUALS red_alert

Do: process_red_alert

Until: status OF red_alert_light EQUALS pr

The mouse is used to position the mouse cursor at the desired scroll arrow and clicked to alter the contents of the window. The content of the window is moved in the opposite direction from the arrow. For example, when the the top scroll arrow is clicked, the contents move down, bringing the view closer to the top of the list or document. Each click of the single arrow moves the window contents one *line* in the chosen direction; each click of the double arrow moves the window contents one *page* in the chosen direction. Continuous depression of the mouse results in continuous movement in the chosen direction. Once the top or bottom of the window contents is reached, depression of the scroll arrows in that direction are ignored.

Dialog Boxes

Dialog boxes are rectangular areas within the HOS window that are used to supply necessary information for a simulation component. Three types of dialog boxes, as described below, are used within HOS: (1) text entry boxes, (2) click boxes, and (3) check boxes.

Text Entry Boxes

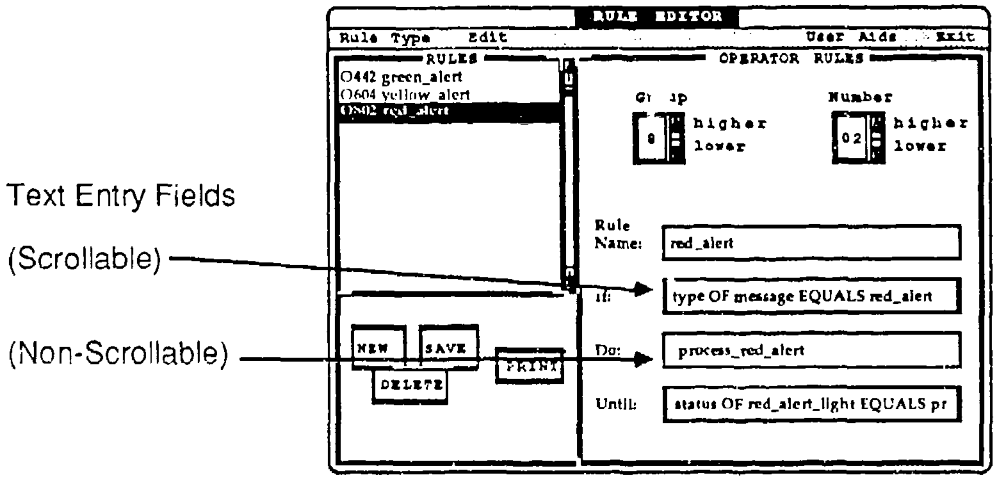
Text entry boxes are fields where textual or numerical data are entered and are illustrated below. Text entry boxes are represented on the screen as rectangular areas drawn with a single blue line. If the entry in the box can be larger than the box size, a double blue line is placed on the left and right to indicate that the user can scroll right and left within this box. The rectangular text cursor is initially placed at the left side of the box. The user can use the mouse or arrow keys (right and left) to position the text cursor to the location where the next keyboard entry is to be placed. All keyboard strokes are inserted at the current location of the text cursor. If the entry causes the length of the current line to exceed the display width, either of the following will occur:

1. If the box is the exact size of the permitted entry (i.e., the right and left side are single lines), a beep will be sounded and future keyboard entries (except backspace and delete) will be ignored until a non-text key is depressed or
2. If the entry can be larger than the box (i.e., the right and left sides are double lines), the text will be scrolled to the left as additional keys are depressed until the maximum field size is reached.

The left and right arrow keys move the cursor one space in the indicated direction within the text entry box. The home key moves the text cursor to the first character in the box; the end key moves the text cursor to the last character in the box.

Text Entry Box Illustration

The screen below contains four text entry boxes for Rule Name, If, Do, and Until. The If and Until boxes are scrollable which means that the entry typed in can exceed the length of the box. The Rule Name and Do boxes are non-scrollable which means that the length of the information entered can not exceed the length of the box.



O442 green_alert
O604 yellow_alert
O802 red_alert

Group: 8
Number: 02

Rule Name: red_alert

If: type OF message EQUALS red_alert

Do: process_red_alert

Until: status OF red_alert_light EQUALS pr

Text Entry Fields

(Scrollable)

(Non-Scrollable)

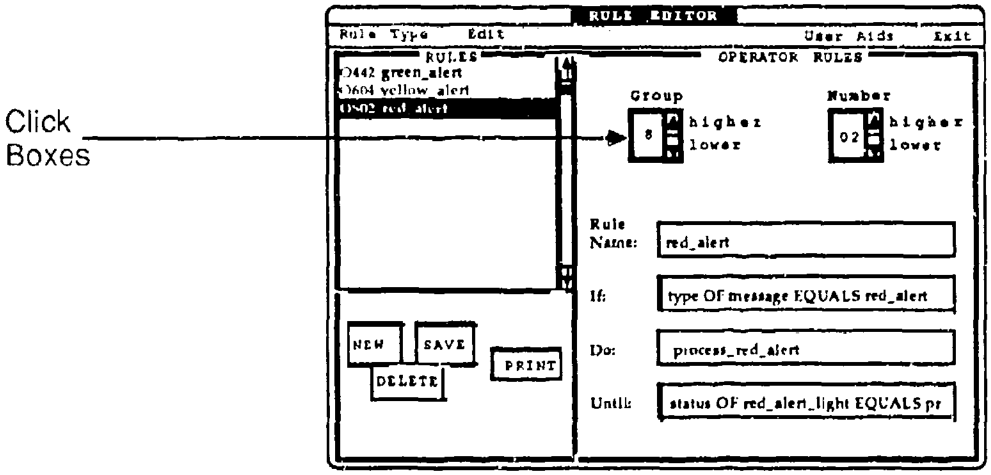
Click Boxes

Click boxes are used to specify numeric values and are illustrated below. The click box is a rectangular area with a box on the left side containing an up and down arrow. The currently di3played number can be modified by:

1. Moving the mouse pointer to the red triangle located above the box to increase the number shown in the box by 1 each time a mouse button is depressed or
2. Moving the mouse po'nter to the red triangle located beneath the box to decrease the number shown in the box by 1 each time a mouse button is depressed.

Click Box Illustration

The screen below contains two click boxes, one for entry of the rule group and one for the rule number. The default value is 0 for both fields. To change the rule group to 10, the up arrow labeled higher would be clicked twice.



0442 green_alert
0604 yellow_alert
0802 red_alert

Group: 8

Number: 02

Rule Name: red_alert

If: type OF message EQUALS red_alert

Do: process_red_alert

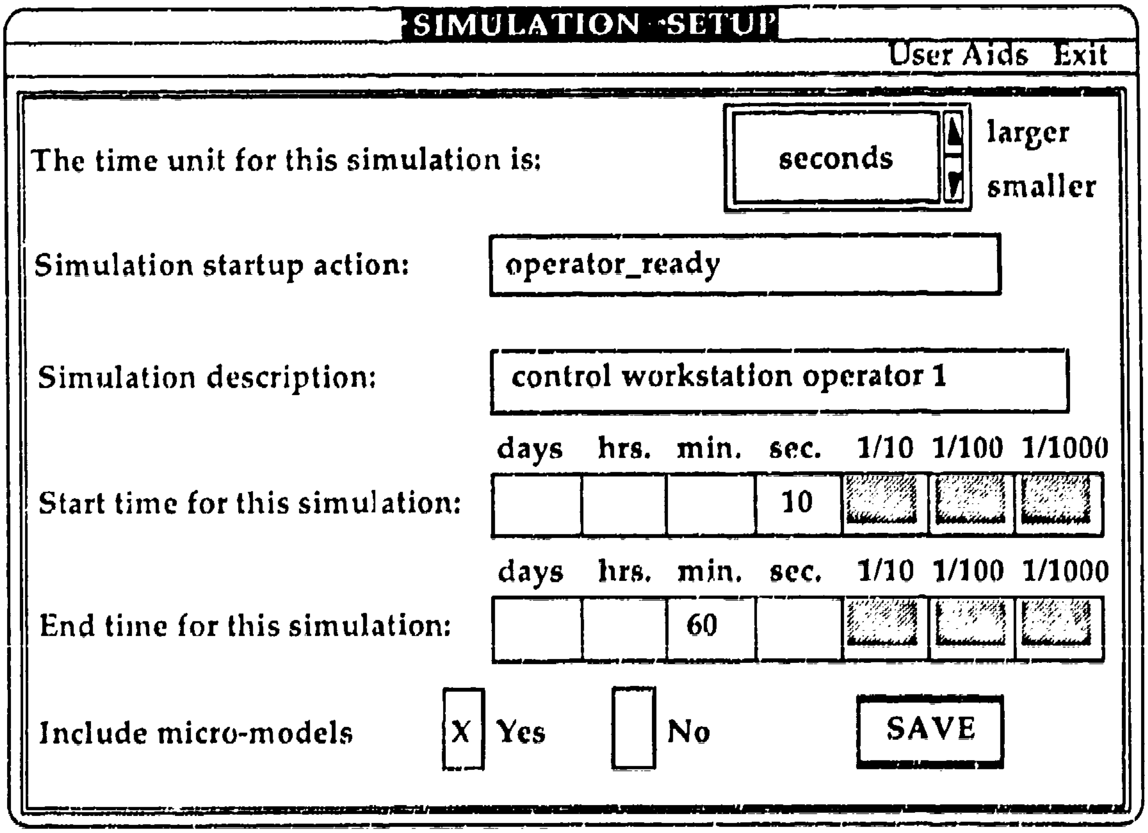
Until: status OF red_alert_light EQUALS pr

Check Boxes

Check boxes are used to select one of a set of options and are illustrated below. The check box is a rectangular area with a descriptive label on the right side. The currently selected option will have an X inside the box. To select a different option, move the mouse pointer so that it is located within the check box for the desired option and click. The X will appear in the selected box.

Check Box Illustration

The screen below contains a check box to indicate whether the HOS micromodels are to be included in the simulation. The X initially appears in the 'Yes' box to indicate that the micromodels are to be included. If the micromodels are not to be included, the mouse would be used to move the mouse cursor to within the 'No' box and clicked. The X will then appear in the 'No' box.



The time unit for this simulation is: seconds

Simulation startup action: operator_ready

Simulation description: control workstation operator 1

	days	hrs.	min.	sec.	1/10	1/100	1/1000
Start time for this simulation:				10	▨	▨	▨
End time for this simulation:			60		▨	▨	▨

Include micro-models ☒ Yes □ No

List Boxes

List windows contain lists of items that the user can view or select. HOS uses two types of list windows: List Selection and List Viewing that are described below.

List Selection Box

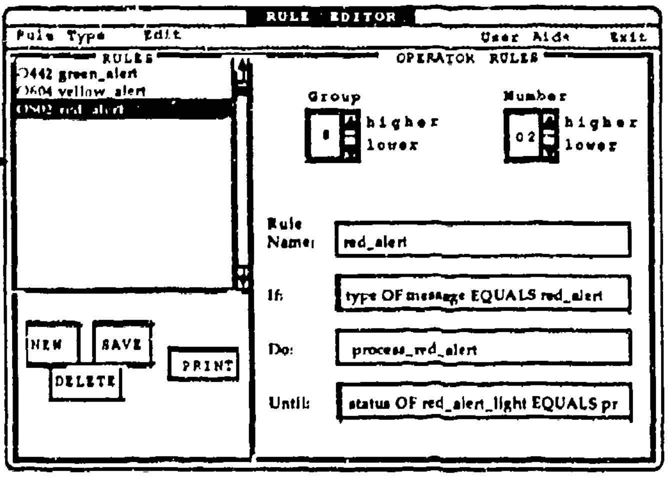
A list selection box presents a list of all items available for the current function, e.g., list of rules for the rule editor, actions for the action editor, objects for the object editor, etc. It has a scroll bar on the right side of the window to be used to alter the viewing area of the window. The list box window may obscure the previous contents of the screen while it is active. It is illustrated below.

The first item in the window is initially selected. The mouse is used to move the mouse cursor so as to point to the name of the item to be selected. The currently selected item is shown in reverse video. If the desired item is not currently displayed within the window, the user can move the mouse cursor to the red triangles located at either end of the scroll bar and then depress the mouse button. Each click on the red triangle will display the next set of items in the window. If the user clicks on the up (down) triangle and the pointer is already located at the first (last) item, the contents of the screen will remain identical. Once the desired item is selected, the mouse cursor must be moved to the appropriate pushbutton to invoke the desired action.

List Selection Box Illustration

The screen shown below contains a list selection box to view the list of the currently defined rules. The user has currently moved the mouse cursor over the rule named red_alert and this selected rule is highlighted.

List Selection Box



0442 green_alert
0604 yellow_alert
0802 red_alert

Number: 02

Rule Name: red_alert

If: type OF message EQUALS red_alert

Do: process_red_alert

Until: status OF red_alert_light EQUALS pr

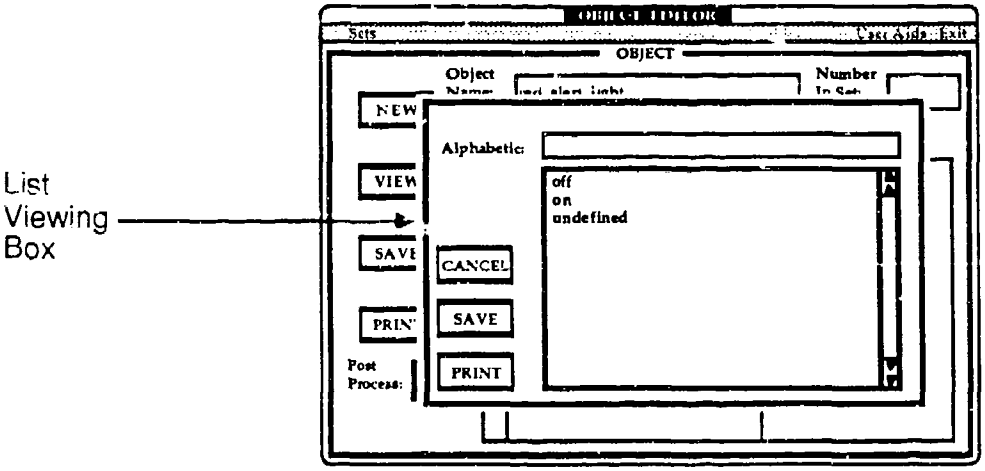
List Viewing Box

The list viewing box displays a list of all defined items for the user to view, e.g., list of alphabetics for the object editor, in a separate rectangular window. It has a scroll bar on the right side of the window to be used to alter the viewing area of the window. The list viewing box window may obscure the previous contents of the screen while it is active.

The mouse pointer is initially located on the first item in the window. If the desired item is not currently displayed within the window, the user can move the mouse cursor to the red triangles located at either end of the scroll bar and then depress the mouse button. Each click on the red triangle will display the next set of items in the window. If the user clicks on the up (down) triangle and the pointer is already located at the first (last) item, the contents of the screen will remain identical. Once the viewing of the defined items is complete, the mouse cursor must be moved to the appropriate pushbutton to invoke the desired action.

List Viewing Box Illustration

The screen below shows a list viewing box that contains the list of currently defined alphabetics.



Object Name: red alert light

Number In Set:

Post Process:

Alphabetic:

off
on
undefined

4. GETTING STARTED

BEFORE YOU BEGIN

This chapter should be read carefully before using HOS. It provides information on the computer hardware and software that is needed to run HOS. It also contains instructions on how to install HOS on your computer.

HARDWARE REQUIREMENTS

To use HOS-IV, an IBM PC/AT or 100 percent compatible is required. The following minimum configuration is recommended:

- An Enhanced Graphics (EGA) monitor,
- An Enhanced Graphics (EGA) card with 256 Kilobytes (Kb) of RAM,
- At least one 5 1/4" floppy diskette drive able to read 360 Kb formatted floppy diskettes,
- A hard disk with at least 10 Megabyte (Mb) of available storage,
- A minimum of 640 Kilobytes (Kb) of memory (RAM),
- A minimum of 1 Megabyte (Mb) of extended RAM that conforms to the EMS specification and functions as a RAM drive, and
- A mouse.

The optimal configuration for using HOS-IV requires the following components:

- 80287 math co-processor,
- A total of 4 Mb of RAM,
- Epson compatible dot matrix printer with graphics capability, and
- 40 Mb Bernoulli Box.

SOFTWARE REQUIREMENTS

DOS 3.2 or a higher version of the operating system is recommended for HOS-IV. **Although HOS-IV can be run using DOS 3.1, certain problems with DOS 3.1 cause HOS-IV to abnormally terminate as will be described in more detail in the section labeled HOS modules.** In addition to the HOS-IV software, a copy of the Microsoft C Version 4.0 compiler and linker is required.

WHAT IS INCLUDED ON THE HOS DISKETTES

HOS software is delivered as one of the following versions:

1. Version A contains all of the HOS software and the Microsoft C V4.0 libraries and
2. Version B contains only the HOS software.

Version A is only sent to registered owners of Microsoft C V4.0 software. Recipients of Version B will have to install their copy of Microsoft C in order to execute HOS. General instructions for installing Microsoft C as required for HOS are contained in Appendix A.

The HOS software is contained on several (n) 360 Kb double-sided diskettes (the number of disks is dependent upon the HOS version, A or B). The following diskettes are needed to install HOS:

1. The HOS utility diskette containing the HOS installation procedure and miscellaneous utility (these utility files are described in more detail in Appendix A) and
2. A set of n diskettes that contain the HOS software for the appropriate version (A or B) including the HOS user aids and sample simulations.

RADIO and SAMPLER are sample simulations that will be used in this document to illustrate how to use HOS.

THE ENTER KEY

The Enter or Return key is usually labeled with a bent left arrow (↵) on the keyboard and is used to indicate the end of a line. Generally, the operating system (or DOS) will not process anything that has been typed until the Enter key is pressed. All the examples in this section will remind the reader that the Enter key must be depressed at the end of each DOS command.

HOS SOFTWARE INSTALLATION

HOS requires a minimum of 10 megabytes of storage. Therefore, at least this amount of space must be free on the storage device. HOS includes provisions for installation on either a hard disk or Bernoulli box as described below.

INSTALLATION PROCEDURE

These instructions assume the the mouse driver has been installed and that the mouse driver files are specified in the file named AUTOEXEC.BAT. In addition, the Ramdrive device driver associated with the extended RAM must be installed. For general details on the procedure to install the mouse and Ramdrive, consult Appendix A. Procedures for installing HOS both on a typical computer equipped with a least one diskette drive (drive a) and a hard disk drive (drive c) and on a computer equipped with auxiliary storage such as a Bernoulli box are described below. Before installing HOS, it is recommended that the file named README.DOC on the utilities diskette be read. The installation procedures will request that the diskettes labeled HOS BACKUP DISK m OF n be inserted in the diskette drive. Make sure to enter the disks in the correct sequence. The installation procedure halts after the prompt until any key on the keyboard is depressed to indicate that the diskette has been inserted in the drive and is ready to be processed.

Installation on a Floppy Diskette and Hard Drive System

To install the HOS-IV software on a computer equipped with a floppy diskette drive configured as drive a and a hard disk drive configured as drive c, perform the following steps:

1. Power up the computer.
2. Wait until the operating system prompt (usually a C>) is displayed (NOTE: this procedure assumes that you will be in the root level of the directory on the C drive; if an autoexec.bat files that is automatically executed contains commands that change to a subdirectory, enter the command **cd** \ to return to the root level).
3. Insert the diskette labeled 'Utility Disk' into the 5 1/4" floppy diskette drive able to read 360 Kb diskettes.
4. Type **a:install** and then depress **Enter**.
5. Follow the instructions displayed on the screen and insert each disk as requested. **Note:** the screen prompts will request insertion of backup diskettes 1 through n; use the HOS diskettes labeled HOS Backup Disks 1 through n.
6. Type **cd \hosiv** and then depress **Enter**.
7. Type **setupsys** and then depress **Enter**.
8. The prompt "Are you using an external disk device, such as a Bernoulli box (Y/N)" requests information regarding the type of storage device that will be used. Since the Bernoulli box will not be used, type an N in response to the prompt.
9. The prompt "Are you using a bus mouse or a serial mouse (B/S)" allows you to specify the type of mouse on your system. A response of B indicates a bus mouse (i.e., one that is attached to an internal card); a response of S indicates a serial mouse (i.e., one that is attached to a serial port on your computer).

The installation process is illustrated below. The user responses are shown in bold typeface.

HOS Installation Screen for a Floppy Diskette and Hard Drive System

```
C > a:install

Insert backup diskette 01 in drive A:
Strike any key when ready

•  •  •

Insert backup diskette 20 in drive A:
Strike any key when ready

C> CD\HOSIV
C >setupsys
Are you using an external disk device,
 such as a Bernoulli box (Y/N)n
Are you using a bus mouse
 or a serial mouse (B/S)s

C>
```

Installation on a Floppy Diskette and Auxiliary Drive System

To install the HOS-IV software on a computer equipped with a floppy diskette drive configured as drive a and an auxiliary drive, such as a Bernoulli Box configured as drive e or f, perform the following steps:

1. Power up the computer.
2. Wait until the operating system prompt (usually a A>) is displayed (NOTE: this procedure assumes that you will be in the root level of the directory on the auxiliary drive; if an autoexec.bat files that is automatically executed contains commands that change to a subdirectory, enter the command **cd** \ to return to the root level).
3. Change the default drive to the one where HOS is to be installed, e.g., e: for drive e or f: for drive f.
4. Insert the diskette labeled 'Utility Disk' into the 5 1/4" floppy diskette drive able to read 360 Kb diskettes.
5. Type **a:install** and then depress **Enter**.
6. Follow the instructions displayed on the screen and insert each disk as requested. **Note:** the screen prompts will request insertion of backup diskettes 1 through n; use the HOS diskettes labeled HOS Backup Disks 1 through n.
7. Type **cd \hosiv** and then depress **Enter.**
8. Type **setupsys** and then depress **Enter.**
9. The prompt "Are you using an external disk device, such as a Bernoulli box (Y/N)" requests information regarding the type of storage device that will be used. Since the Bernoulli box already contains HOS, type an N in response to the prompt.

10. The prompt "Are you using a bus mouse or a serial mouse (B/S)" allows you to specify the type of mouse on your system. A response of B indicates a bus mouse (i.e., one that is attached to an internal card); a response of S indicates a serial mouse (i.e., one that is attached to a serial port on your computer).

The installation process is illustrated below. The user responses are shown in bold typeface.

HOS Installation Screen for an Auxiliary Drive System

```
E >a:install
HOS IV initial installation process has begun...
The HOS IV environment has been created
The HOS IV system will now be loaded...

Insert backup diskette 01 in drive A:
Strike any key when ready

Insert restore diskette 01 in drive E:
Strike any key when ready

•  •  •

Insert backup diskette 20 in drive A:
Strike any key when ready

E> CD\HOSIV
E> setupsys
Are you using an external disk device,
 such as a Bernoulli box (Y/N)n
Are you using a bus mouse
 or a serial mouse (B/S) s

E>
```

MAKING BACKUP COPIES

The HOS software is not copy protected. After installing the HOS software, at least one backup copy of of the HOS software should be made. In addition, the backup procedure should also be performed periodically during the development of any simulation. The DOS **backup** command can be used to copy all software associated with HOS to floppy diskettes. Formatted diskettes are required to execute the backup procedure. To backup all the HOS and simulation files, perform the following steps:

1. Power up the computer if it's not already on.
2. Wait until the operating system prompt (usually a C>) is displayed.
3. Type **backup \hosiv a: /s** and then depress **Enter**. The **a:** in the backup command contains the letter of the drive that receives the backup files. **a:** is generally the drive designator for the floppy diskette drive.
4. DOS will generate a beep and then display the following warning:

   ```
   Insert backup diskette 01 in drive A:
   Warning! Diskette files will be erased
   Strike any key when ready
   ```

   DOS will erase the contents of the diskette so be sure to use blank diskettes or those that do not contain any needed files. The formatted blank diskettes are to be put in the appropriate disk drive and then any key should be depressed. DOS will display the name of each file as it makes the copies.

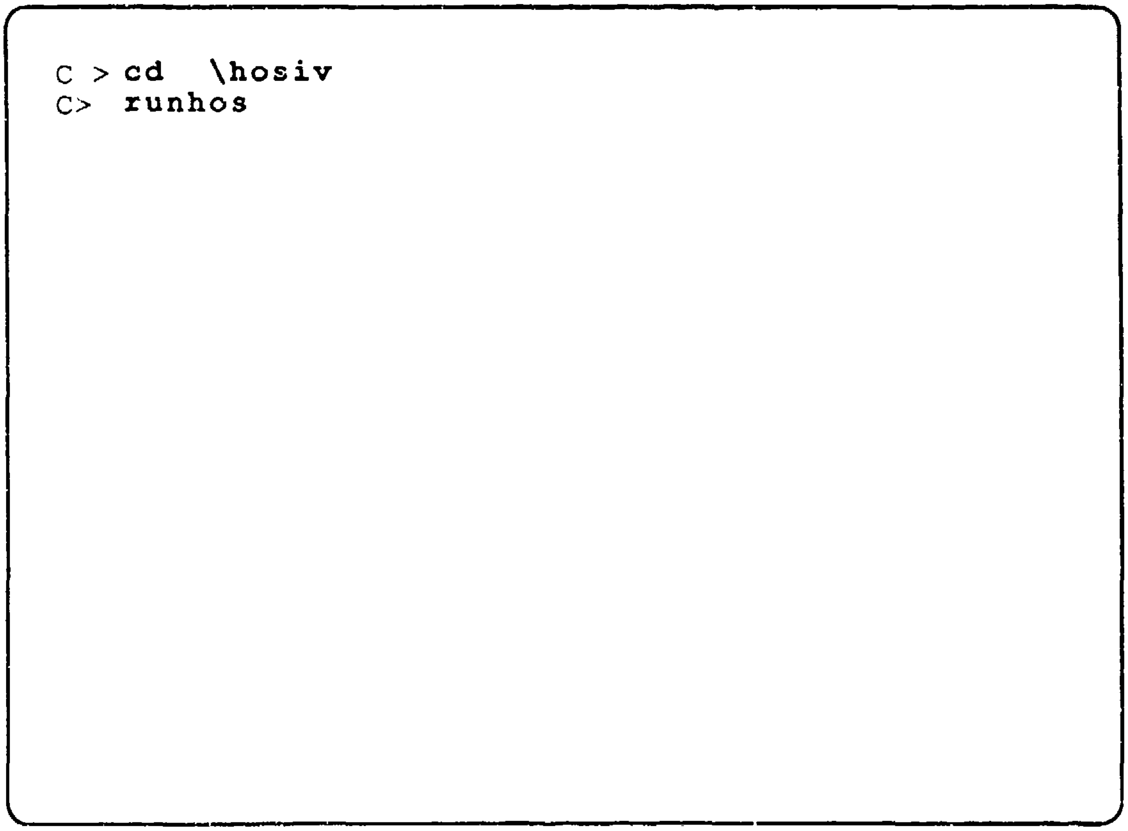
After installing the HOS software, store the original disks in a safe place. That way, should anything happen to your working copy, the original disks will still be intact.

HOW TO START HOS

The procedure to start HOS are outlined in the following sections. The instructions include the actual commands as well as brief descriptions. Illustrations of screens are also included to enhance the step-by-step instructions. The operating system prompts the user with a C>. To start HOS:

1. Enter **cd \hosiv** and then depress **ENTER.**
2. After the C>HOSIV prompt, enter **runhos** and **ENTER.**

HOS Start-up Screen

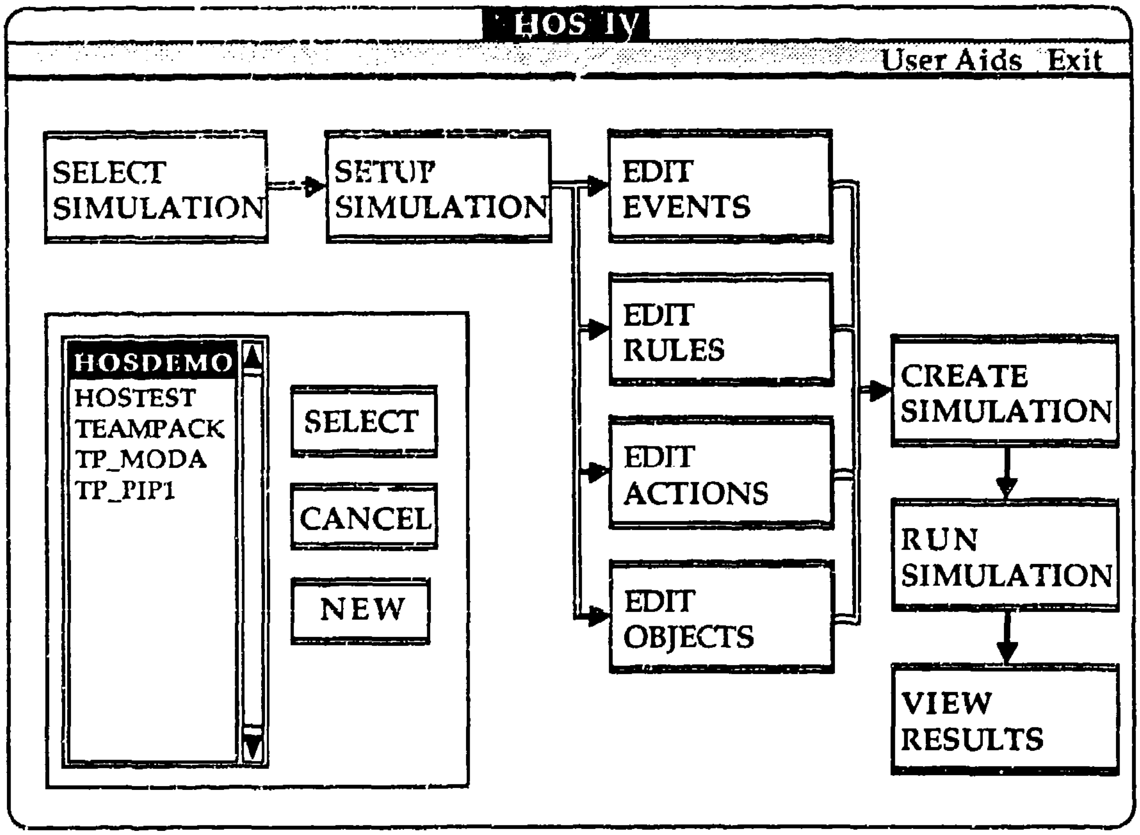


```
C >cd  \hosiv
C> runhos
```

In a few seconds, the HOS main screen will be displayed.

HOS Main Screen

The HOS main screen shows the user the complete set of steps that must be completed. The following sections will describe each step in the process of setting up, executing, and analyzing a HOS simulation. Specifically, each main selection, or module, is described in detail. Sample screens will be used to further illustrate the instructions.



HOS-IV MODULES

The HOS-IV software is split into nine modules which can be executed from the main screen of HOS-IV. The organization of the HOS software is reflected in the flow shown on the HOS main screen. The left to right progression shows the sequence in which the modules must be invoked to define and execute a simulation. As shown on the HOS main screen, these modules are selected by moving the mouse over the title of the selected pushbutton and clicking the mouse.

The HOS-IV functional modules are listed below with the pushbutton label shown in bold:

- **SELECT SIMULATION** — is used to define a new simulation or determine which of the existing simulations is to be used during the HOS simulation (described in Section 5).
- **SETUP SIMULATION** — is used to define and modify basic simulation parameters such as simulation name, time units, simulation start and end times (described in Section 5).
- **EDIT EVENTS** — is used to maintain event information for a simulation including creating new events, modifying existing events, and deleting events (described in Section 6).
- **EDIT RULES** — is used to maintain rule information for a simulation including creating new rules, modifying existing rules, and deleting rules (described in Section 6).
- **EDIT ACTIONS** — is used to maintain action information including creating new actions, modifying existing actions, and deleting actions. In addition, the action editor invokes the action translator that translates an action into C code and

determines if the action contains any errors (described in Section 6).

- **EDIT OBJECTS** — is used to maintain object information including creating new objects, modifying objects, and deleting objects (described in Section 6).
- **CREATE SIMULATION** — integrates all the events, rules, actions, and objects for a simulation and builds the simulation file for subsequent execution (described in Section 9).
- **RUN SIMULATION** — executes the simulation and creates files for simulation analysis (described in Section 9).
- **VIEW RESULTS** — generates simulation analysis reports (described in Section 10).

HOS-IV automatically initiates the Select Simulation module at the beginning of each HOS session. After the user has successfully completed the Select Simulation step, any other module may be selected in any order. Whenever the user exits a module, the HOS-IV main menu returns. The user can select the next module or terminate HOS-IV. The sequence of the remainder of the modules is user-driven.

DOS 3.1 Warning

If DOS Version 3.1 is being used as the operating system, HOS may abnormally terminate and return to the system when the user terminates one of the HOS modules (the user will see the C>HOSIV prompt). All processing performed for the module is successfully saved. To recover from this abnormal termination, restart HOS be entering **runhos**. The HOS main screen will be displayed. Select the desired simulation and then the desired module. Processing will continue normally at this point.

HOS-IV Screens Formats

The format for each HOS module remains consistent across all modules. The following sections will briefly describe each component of the format and explain how the user interacts with HOS.

HOS-IV Title Bar

The title bar of HOS-IV contains the words 'HOS IV' as the function name. During the simulation setup phase, the user can enter a unique name to differentiate various simulations as described in Section 5.

HOS-IV Menu Bar

The HOS-IV main menu contains the following menu options on the menu bar:

- **User Aids** — provides the capabilities to view help files. This pull down menu contains Help commands that allow the user to obtain help windows about a HOS-IV module.
- **Exit** — terminates HOS-IV and returns the user to the operating system. Before Exist is selected, the user should SAVE all entries.

HOS-IV Pushbuttons

The HOS-IV main screen contains the following pushbuttons:

- **Select Simulation** — starts the select simulation module.
- **Setup Simulation** — starts the setup simulation module.
- **Edit Event** — starts the Event Editor.
- **Edit Rule** — starts the Rule Editor.
- **Edit Action** — starts the Action Editor.
- **Edit Object** — starts the Object Editor.
- **Create Simulation** — starts the create simulation module.
- **Run Simulation** — starts the simulation.

- **View Results** — starts the view results module.

HOS-IV CONVENTIONS

Certain naming conventions are used throughout the HOS modules and are described in the following sections.

Simulation Names

The name of a simulation can contain a maximum of eight alphanumeric characters. The simulation name cannot contain any special symbols except an underscore (_). Examples of valid simulation names are:

my_sim,

teampack, and

rev_1a.

Examples of invalid simulation names are:

teampack1 Contains more than 8 characters,

team pip Contains an invalid character (space), and

team-pip Contains an invalid character (dash).

Other Names

All other names, such as rule names, object names, etc., can contain a maximum of 28 alphanumeric characters (a-z, 0-9). The first letter of each name must be a character (a-z). The only special symbol that can be included in a name is an underscore (_). HOS is also case insensitive, that is upper and lower case characters can be used interchangeably. All names are converted by HOS into lower case for internal use. Examples of valid names are:

myname

my_sim_variable_name

abc123456

Examples of invalid names are:

a$	Contains an invalid special symbol ($),
a b c	Contains spaces,
a-b	Contains an invalid special symbol (-),
1qwerty	First letter is not a character.

5. DEFINING A SIMULATION

This section describes how to use the Selection Simulation module to create a new simulation and access a previously created simulation. It also provides information about how to use the Setup Simulation module to define the basic simulation parameters such as time units, start and end simulation times, etc. The information presented for each module first describes the information required by the module and factors the user should consider in defining the information. Secondly, details on how to enter the information are provided.

SELECT SIMULATION

The **Select Simulation** module determines the simulation to be used and is also used to create new simulations. The Select Simulation module is automatically executed whenever HOS is started since the other modules cannot properly function without knowing which simulation is to be used. In addition, Select Simulation option can be selected whenever the HOS main menu is displayed in order to switch to a different simulation.

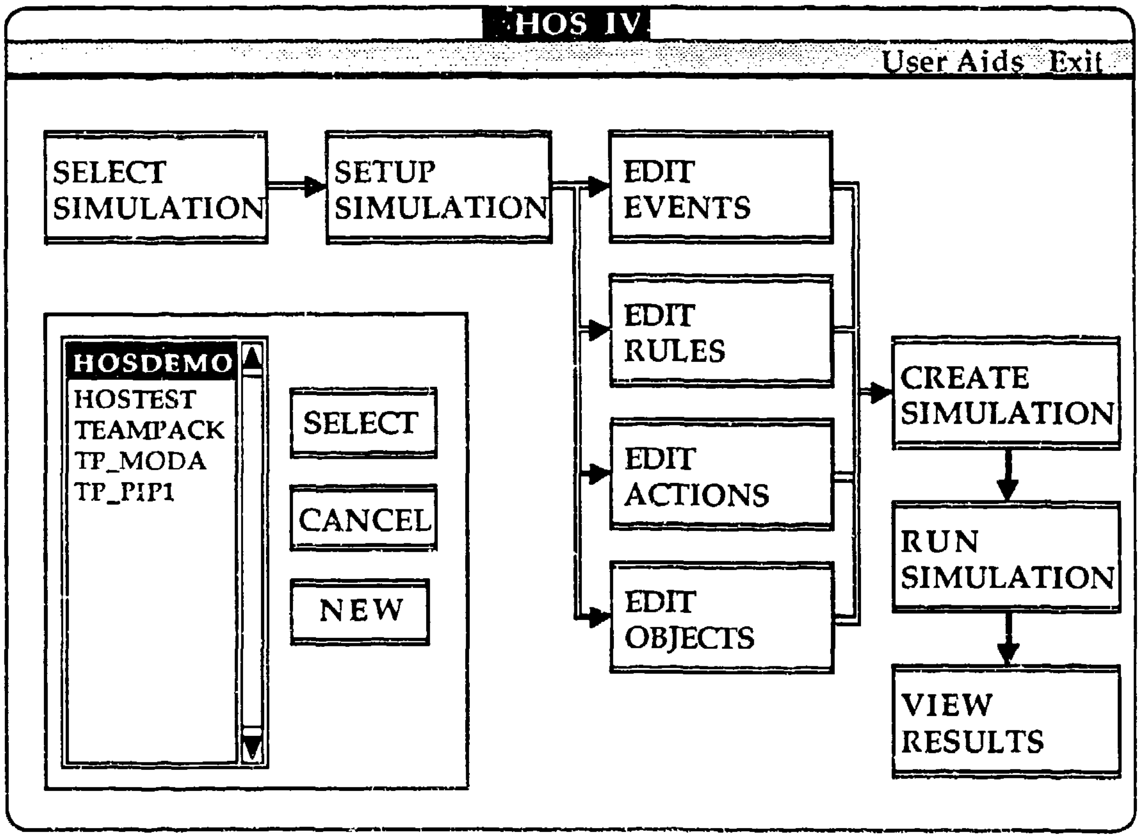
If any simulations exist, Select Simulation displays a list selection box which allows the user to scroll through the list of currently defined simulations and select one. The user can also indicate that a new simulation is to be created or, if desired, cancel the Select function. Whenever the user cancels Select Simulation without selecting a simulation, an informative message is displayed indicating that the HOS session cannot continue until a simulation is selected. The response to this message is to exit (terminate the HOS session) or continue (return to Select Simulation).

Simulation Naming Conventions

The name of a simulation can contain a maximum of eight alphanumeric characters. The simulation name cannot contain any special symbols except an underscore (_).

Select Simulation Screen

The Select Simulation dialog window overlays the HOS main menu and is illustrated below.



Title Bar

The title bar of Select Simulation contains the words 'HOS-IV' as the function name.

Menu Bar

The menu bar is not functional while the Select Simulation dialog window is on the screen.

Select Simulation Windows

The main Select Simulation window is a dialog window for entering object information and is shown above. The list selection box contains the list of currently defined simulation names.

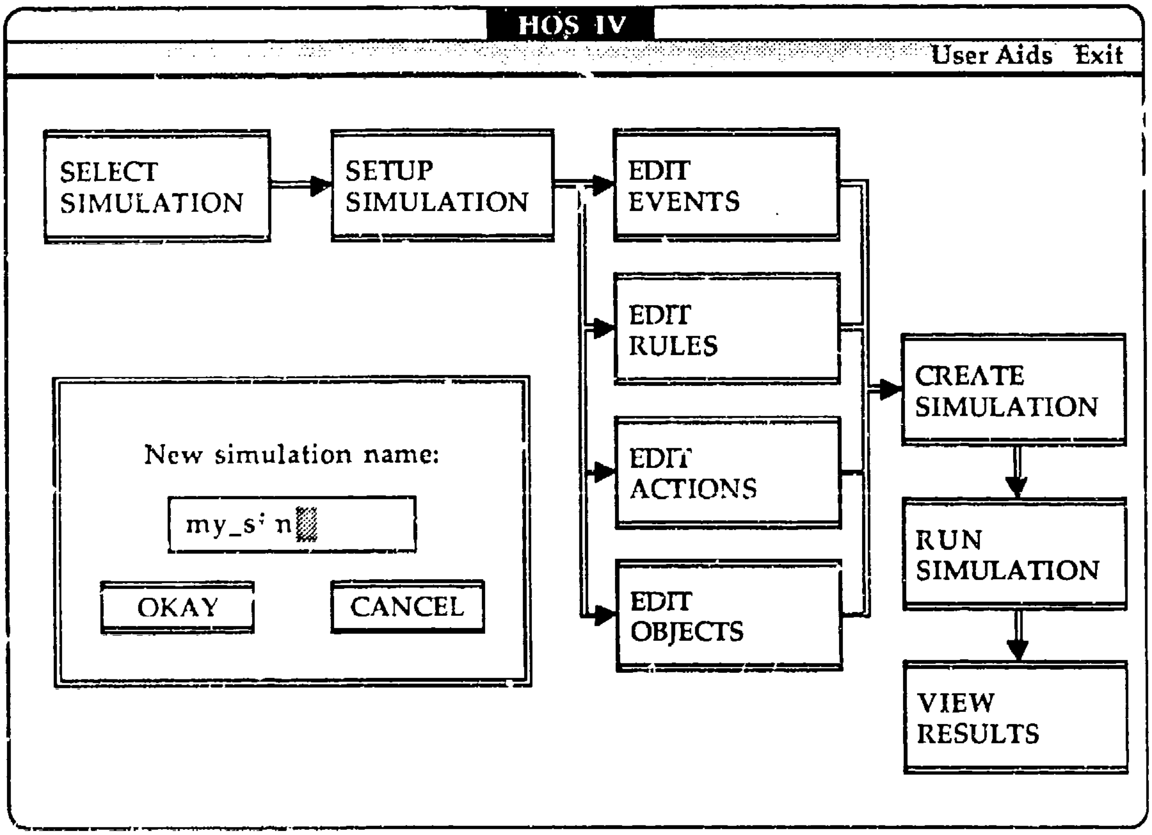
Pushbuttons

The Select Simulation dialog window, as illustrated above, contains the following pushbuttons:

- **SELECT** — the simulation name highlighted in the list selection box is selected as the current simulation.
- **CANCEL** — terminates the Select Simulation function and returns the user to the main HOS-IV menu without changing the current simulation. If no simulation has been selected during the session, a warning message will be displayed.
- **NEW** — indicates that a new simulation is to be defined and generates the new simulation dialog window.

Defining a New Simulation

When the user selects **NEW** from the Select Simulation dialog window, a new simulation window is displayed as shown below.



New simulation name:

my_s' n

The new simulation dialog window contains a text entry box for entry of the simulation name as shown above. The example shows the user entering the name of the new simulation called my_sim.

Pushbuttons

The following pushbuttons are available:

- **CANCEL** — cancels the new simulation function.

- **OKAY** — checks the entered simulation name to ensure that it does not contain any illegal characters and that a simulation with the same name has not been previously defined. If the simulation name is valid, the simulation name is added to the list and the simulation directories are created.

Select Simulation Message Windows

Select Simulation generates the following message windows:

- **No simulation selected** — indicates that a simulation must be selected before HOS-IV can continue. The user options are to select a simulation or exit from HOS-IV.
- **Name already used** — indicates that the simulation name just entered has been previously used for a simulation and a different name must be entered. After reading the error message, the user must hit the **Okay** pushbutton to return to the Define new simulation window.
- **Invalid character** — indicates that the simulation name just entered contains an invalid character and must be re-entered. After reading the error message, the user must select **Okay** to return to the Define new simulation window. This allows the user to define a new name and continue the simulation.

SETUP SIMULATION

The Setup Simulation module of HOS maintains general information needed to run each simulation. When the Setup Simulation module is executed, a dialog window is created with entry fields displayed for each required piece of information. These fields include:

- Time unit — indicates the simulation time unit as: Days, Hours, Minutes, Seconds, Tenths (1/10 seconds), Hundredths (1/100 seconds), or Thousandths (1/1000 seconds). The **default** time unit is seconds.

- Simulation startup action — indicates the action that is to be performed to start the simulation. There is no **default** start up action.
- Simulation description — text containing a brief description (up to 80 characters) of the simulation. There is no default description.
- Simulation start time — the time the simulation is to start. The **default** simulation start time is zero.
- Simulation end time — the time the simulation is to end. The **default** simulation end time is zero.
- Includc micromodels — indicate whether the standard HOS micromodels are to be included in the simulation. The **default** value is yes.

Simulation Time Units

When selecting the simulation time unit, the scroll arrows are used to scroll through a list containing the time unit options until the desired time unit is displayed in the box. To increase the time unit, e.g., from seconds to minutes, the top arrow is used. Similarly to decrease the time unit, e.g., from seconds to tenths of seconds, the bottom arrow is used. To use a scroll arrow, the mouse is used to move the mouse cursor over one of the arrows and the mouse button is clicked.

One of the most important decisions about a simulation is determining the time units to be used. Only one time unit can be specified for a simulation. The processes to be simulated for the environment, hardware, and operator must be evaluated to determine which one occurs at the smallest time interval that is **meaningful** to the simulation. For example, if the smallest process is an operator performing manual tasks, then the time interval may be seconds or minutes. If the smallest process is an operator reading information from a display, then the time interval may be tenths of seconds. Or, if the smallest microprocess is associated with a hardware activity such as a single update for a 60 Hz radar signal detector, then the simulation time unit may be milliseconds. However, in the last case, the use of milliseconds may not yield information that is useful to the simulation at the millisecond level and therefore a larger time unit should be selected. Many times, it is useful to break up the mission into phases with each phase simulated separately so that the smaller time units are used only for those mission phases where the smaller time unit yields useful information.

The selection of the time unit to be used for the simulation also impacts the length of time required to run a HOS simulation and therefore should be selected judiciously. About 1 second of time is required to process each simulation time unit regardless of the actual time unit selected. Therefore a 1 hour simulation will require 1 minute to run if the time units are minutes; 1 hour if the time units are seconds but about 3 hours if the units are hundredths of seconds. Additionally, certain information about the simulation is written to files every simulated time unit. The smaller the time interval, the larger the analysis file, and the possibility of exceeding available disk storage space must also be considered.

Simulation Startup Action

At the beginning of the simulation, HOS firsts executes the action named as the startup action (for more details about actions, see section 6). This startup action typically initializes object values and starts rules for the environment, hardware, and/or operator. The name of the startup action must be supplied by the user. The BEGIN_SIMULATION startup action, included in the HOS sample simulation described in Section 10, represents a simple case in which the action only contains HAL statements that start all the simulation rules. If only the environment and hardware are to be included in the simulation, then only the hardware and environment rules need to be started and all operator activities included in the simulation will be ignored. Similarly, if only the operator is to be simulated, then the startup action need only start the operator rules and all hardware and environment activities will be ignored.

Simulation Description

The simulation description is a short statement that consists of 80 characters and is used to identify the simulation. This description is displayed on the top of all simulation outputs.

Start and End Simulation Times

The simulation start time indicates the time at which the simulation is to start. The default causes the simulation to start at time zero. However, the simulation can start at any time designated by the user. This is a useful feature when only a particular segment of a mission is to be simulated and all earlier times do not have to be simulated. For example, if the first 60 minutes of the simulation represent the time required for the operator to arrive at a duty station and the simulation is not concerned with processes that occur during the transit time, the simulation start time could be set to 60 minutes.

The simulation end time indicates the time at which the simulation is to end. If the simulated operator is on a two hour shift, then the simulation end time should be set to two hours. If the course of events that occur during the simulation determine when the simulation is complete, the end time should be set at a point beyond the expected end time. The capability of ending the simulation when certain conditions occur from within actions is also provided by the HOS software (see END_SIM in Section 7).

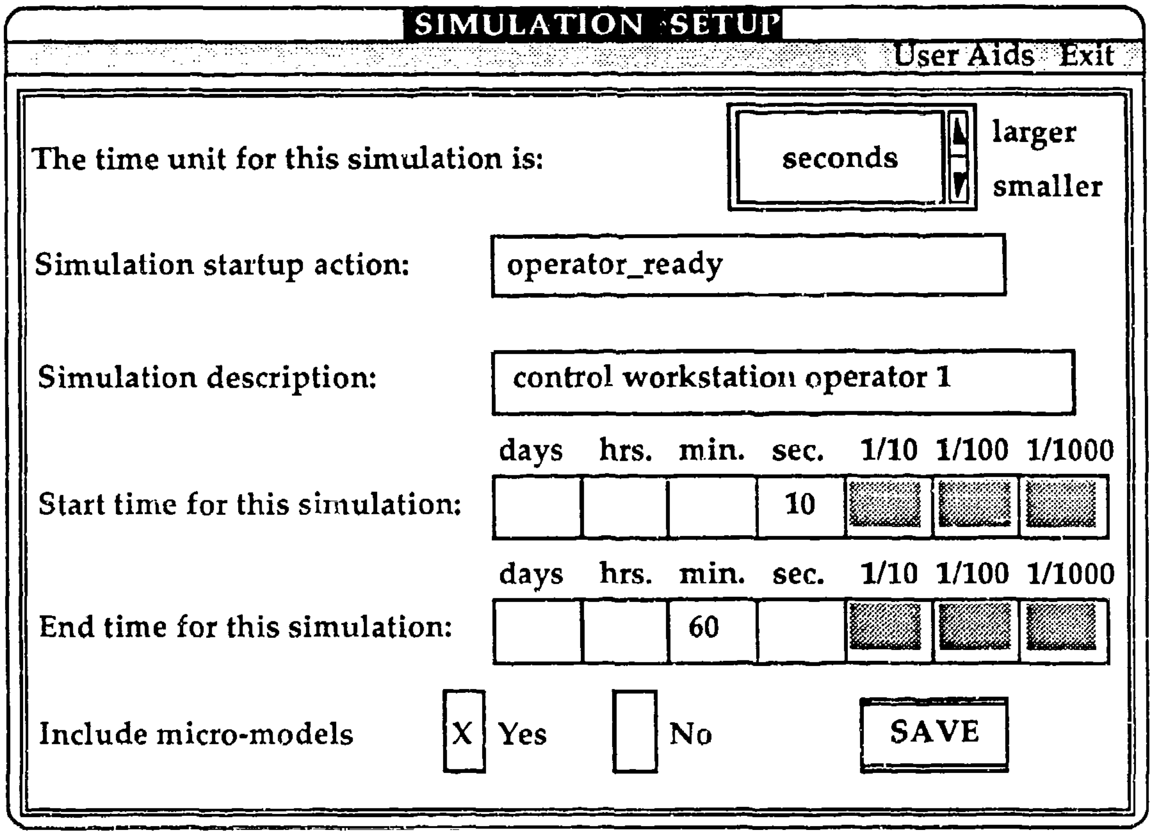
The simulation start and end times consist of seven fields: days, hours, minutes, seconds, tenths of seconds, hundredths of seconds, and thousandths of seconds. Only those fields greater than or equal to the selected time unit can be accessed by the user. Invalid fields are blocked out by a blue box covering the field. This is illustrated in the following **Simulation Setup** screen.

Include Micro-Models

The HOS micromodel libraries (as described in Section 8) are automatically included in all simulations unless the user selects the 'No' check box during the Simulation Setup. It the user does not include the libraries, the Micromodel actions, objects, and rules will not be added to the user's simulation.

Setup Simulation Screen

Each field of the Simulation Setup screen is illustrated below:



The time unit for this simulation is: seconds

Simulation startup action: operator_ready

Simulation description: control workstation operator 1

	days	hrs.	min.	sec.	1/10	1/100	1/1000
Start time for this simulation:				10	▨	▨	▨
End time for this simulation:			60		▨	▨	▨

Include micro-models ☒ Yes ☐ No

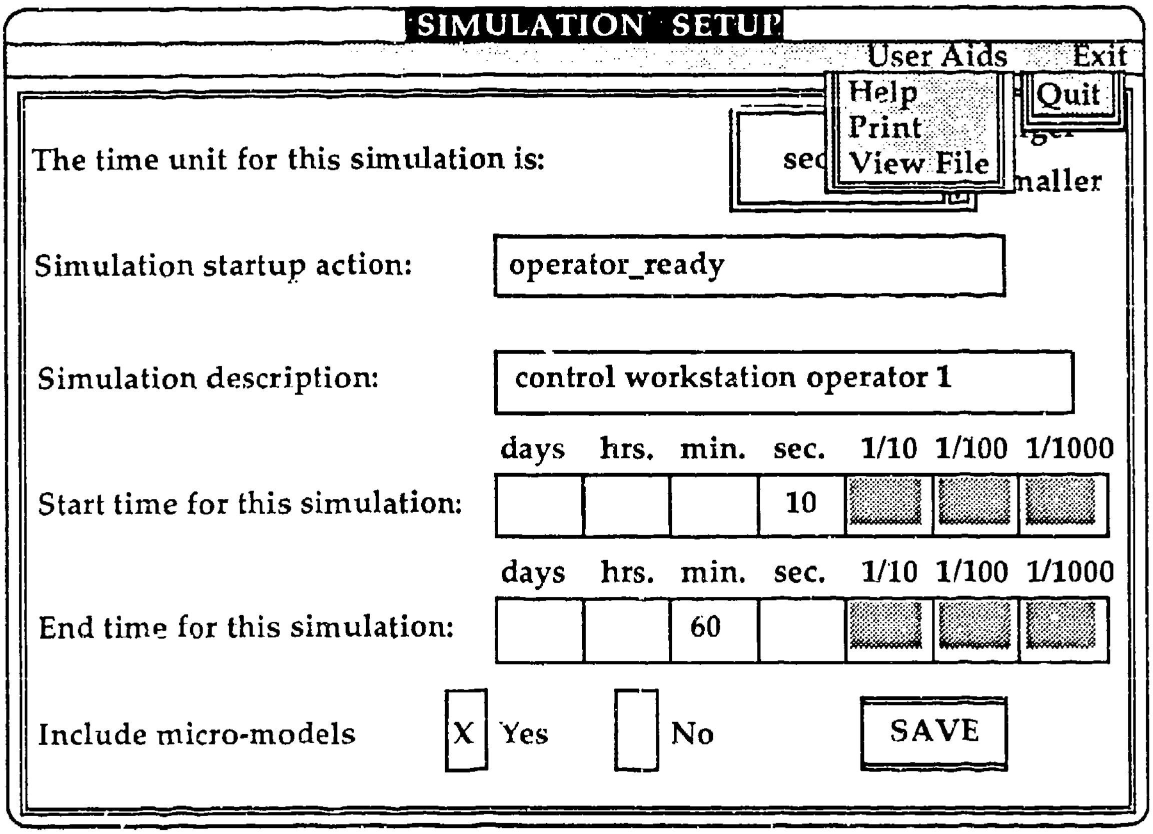
Title Bar

The title bar of Setup Simulation contains the words 'Setup Simulation' as the function name.

Setup Simulation Menu Bar

Setup Simulation screen contains the following menu options on the menu bar:

- **User Aids** and
- **Exit.**



The time unit for this simulation is: sec ... aller

Simulation startup action: operator_ready

Simulation description: control workstation operator 1

	days	hrs.	min.	sec.	1/10	1/100	1/1000
Start time for this simulation:				10	▨	▨	▨
End time for this simulation:			60		▨	▨	▨

Include micro-models ☒ Yes □ No

User Aids Commands

The **User Aids** pull-down menu contains commands that allow the user to view other files, print current setup simulation information, and obtain help messages. It contains the following commands:

- **View Files** — allows the user to obtain a window that displays currently defined actions.
- **Print** — allows the user to obtain a printout of setup simulation information.
- **Help** — allows the user to obtain additional information about using the setup simulation module.

Setup Simulation Windows

The main Setup Simulation window is a dialog window for entering setup simulation information. It contains one option for saving simulation setup information.

Pushbuttons

The Setup Simulation dialog window, illustrated above, contains the following pushbutton:

- **SAVE** — saves the current setup simulation information as displayed on the screen. The following validation is performed:
  1. The start simulation time is compared to the end simulation time. If the start time is greater than the end time, an error message window is displayed indicating that the start time is too large.
  2. The start action is evaluated to determine if the action name is valid and has been defined. If it has not been defined, a warning message window is displayed indicating that the action is undefined.

Text Entry Boxes

The Setup Simulation dialog window contains the following text entry boxes:

- **Startup action** — entry of an action name. In this case, the startup action name is operator_ready.
- **Description** — entry of the simulation description as a maximum of 80 alphanumeric characters including spaces. In the sample above, the simulation description is: control workstation operator 1.
- **Start Simulation Time** — entry of the start simulation time in the form of seven two-digit numbers, one for each time unit. In the example above, the simulation will start at 10 seconds.

- **End Simulation Time** — entry of the maximum simulation time in the form of seven two-digit numbers, one for each time unit. In the sample above, the simulation will terminate at 60 seconds.

List Selection Box

A list selection box showing the selected simulation time unit is displayed in the upper right corner of the Setup Simulation dialog window. To change the time unit to a larger unit, e.g, hours, use the mouse to move the cursor to the arrow labeled 'larger' and click twice. Each click will result in the next time unit being displayed in the box. Similarly, to change the time units to hundredths of seconds, use the mouse to move the mouse cursor to the arrow labeled 'smaller' and click twice. The default time unit is seconds.

Check Box

The Setup Simulation dialog window contains a check box to indicate whether the HOS micromodel libraries are to be included in the simulation. The default value of 'Yes' is automatically selected. If the micromodels are **not** to be included, the 'No' option should be selected.

Setup Simulation Message Windows

Setup Simulation generates the following message windows:

- **The start time, 00 00:00:00.000, is greater than the end simulation time** — informs the user that the entered simulation start time is greater than the end simulation time which would terminate the simulation as soon as it began. The user must modify either the start or end simulation time.
- **The action ________ has not been defined** — informs the user that the action named as the startup action has not been defined. The user must subsequently

define the action in order to create the simulation.

6. USING THE HOS EDITORS

This section describes how to use the HOS editor modules to create and maintain object, event, rule, and action information. Brief summaries of the definitions of objects, events, rules, and actions are provided in Section 2 . The information presented for each module first describes the information required by the module and factors the user should consider in defining the information. Secondly, details on how to enter the information are provided.

OBJECT EDITOR

The Object Editor maintains all object information. Objects represent all knowledge about things to be included in a simulation (e.g., displays, controls). A list of the distinctive features of the object (e.g., size, color, location, etc.) that are needed by the simulation are also specified as a list of characteristics and each characteristic is assigned a value. Values indicate the state of the characteristic at a particular point in the simulation, e.g., is the status (characteristic) of a control (object) on or off (value).

The objects can be used to represent whatever level of detail is necessary for a simulation. For example, the operator can be represented as an object with right and left hands as characteristics. If an anatomy movement model required specific information for a hand such as reaching for a specific location in the workspace, then objects may be necessary for each hand. The model can contain extensive detail, such as that necessary to represent fine motor activities in activating a dial or switch, so that each finger of the hand required definition as an object.

Typically, an object is defined for each essential component of the system, e.g., keyboard, display, mouse, etc. All items in the environment that will be simulated will also require object definitions, e.g., emitters, OPFOR systems and other systems. An operator object is also required but a standard definition is provided as part of the HOS micromodels. Objects also can be used to collect measures of effectiveness, e.g., number of emitters processed per hour, number of errors, etc. Again, it is up to the user to decide if a single object representing the keyboard is suitable for a particular simulation or if each key on the keyboard should be a separate object.

Objects are stored in a library that is available to all simulations. This object library is shared by multiple simulations and is **not** simulation dependent. Whenever an object is used in a rule or action, the current object definition is retrieved from the object library. Therefore, it is **not** necessary to redefine an object if it had previously been identically specified for another simulation. That is, the object definition for a pushbutton, display, etc. need only be specified once and can be used by all simulations on the computer.

Object Characteristics

As described above, characteristics of objects represent features that are relevant to a simulation. The type of information contained in characteristics are **whole, decimal**, or **alphabetic**. Whole and decimal represent numeric values that can be used in calculations. **Whole** numbers do not contain a decimal point and can contain only the digits 0-9 and an optional preceding plus (+) or minus (-) sign. Whole numbers can contain values between -2,147,483,648 and2,147,483,647. **Decimal** numbers can contain only the digits 0-9, a single decimal point (.), and an optional preceding plus (+) or minus (-) sign. The valid range of values for decimals is approximately 1.7E-308 to 1.7E+308. **Alphabetics** represent words or text, e.g., on, off, red, average, etc. Alphabetics cannot be used in formulas or calculations. Alphabetic values are strings of up to 28 characters that can contain any symbol except space, single quote, or double quote. The text is entered without any special enclosing characters such as quotes. Similar to the way objects are stored in a central library, alphabetics are maintained in a dictionary that is accessible to all simulations.

An object definition consists of the following information:

1. **Object Name** — unique identifier of up to 28 characters. The first character must be an alphabetic (a-z) and the remainder can contain alphabetics (a-z), numbers (0-9), and underscore (_); and
2. **Characteristic List** — list of characteristic names with associated type and initial value. A maximum of 15 characteristics can be defined for an object. The characteristic name can be a maximum of 28 characters including alphabetics (a-z), numbers (0-9), and underscore (_).

The characteristic type is automatically assigned based upon the contents of the initial value according to the following rules:

1. If the value is a number (digits 0-9) without a decimal point, then type whole is assigned.
2. If the value is a number containing a decimal point, then type decimal is assigned.
3. If the value is neither whole or decimal, then the Object Editor assumes that it is an alphabetic value. The Object Editor will check the dictionary to see if the value has

previously been entered. If not, an error message will be generated.

Object Sets

Objects can either be defined as:

1. Simple, singular objects such as a single display or
2. Sets of objects which represent multiple occurrences of identically defined objects such as a list of 10 messages or a set of 500 emitters.

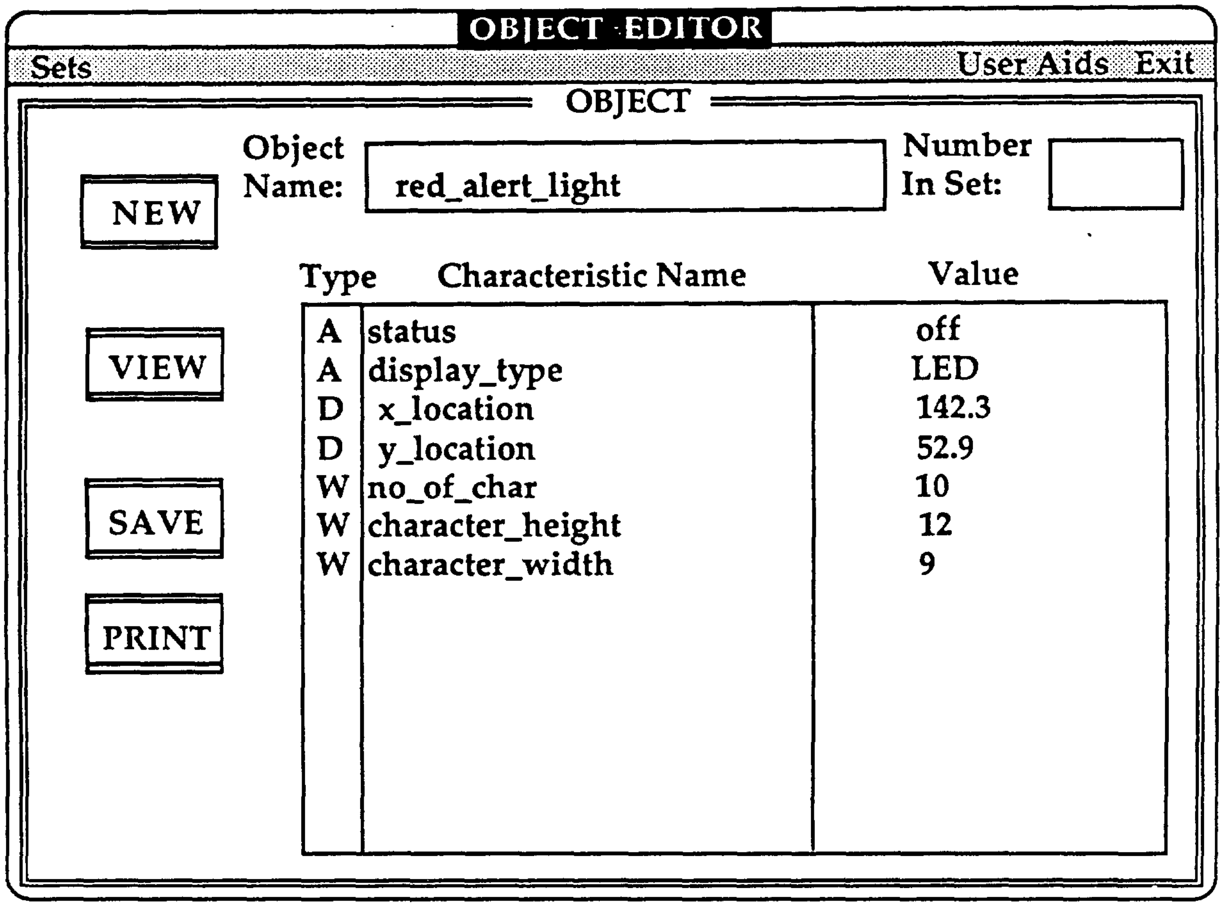
For object sets, the characteristic list must be identical but the characteristic values can vary. The names of the members in an object set are automatically constructed by the Object Editor. The Object Editor appends a sequential number, starting with one and continuing to the end of the object set name. For example, the object set name EMITTER would contain objects named EMITTER001, EMITTER002, EMITTER003, etc. Once a set has been defined, the number of members in the set cannot be changed, i.e., objects cannot be added or deleted from the set nor can any characteristics be modified without redefining the entire set. The only item that can be modified by the user is the value of a characteristic. When sets are created, all members of the set have the initial value entered when defining the set name. For example, if the status (characteristic) of EMITTER was set to off, then all members of the emitter object set will have initial values of off. The Object Editor can be used to access specific members of the object set to change the initial values assigned to the set. For example, if EMITTER 010 is to be on, then the Object Editor should be used to view EMITTER 010, the value for status should be changed to on, and the object definition saved.

Object Libraries

Since the number of objects defined can get rather large, HOS contains a provision to store object information in separate object libraries. For example, the objects for simulation A can be stored in a separate library from the objects required for simulation B. This feature is particularly useful if several simulations are being developed with each simulation utilizing different groups of objects. If the objects required for each simulation are stored in the default object library, information about all the objects will be maintained during the simulation and reported by View Results (see Section 10). Maintaining all the object information can make significant demands both on the amount of disk and file storage needed and the time required to run the simulation. Therefore, HOS provides the ability to name object libraries and indicate which specific library is to be used for a simulation.

Object Editor Screens

The Object Editor screen is illustrated below:



Object Name: red_alert_light

Number In Set:

Type	Characteristic Name	Value
A	status	off
A	display_type	LED
D	x_location	142.3
D	y_location	52.9
W	no_of_char	10
W	character_height	12
W	character_width	9

Title Bar

The title bar of Object Editor contains the words 'OBJECT EDITOR' as the function name.

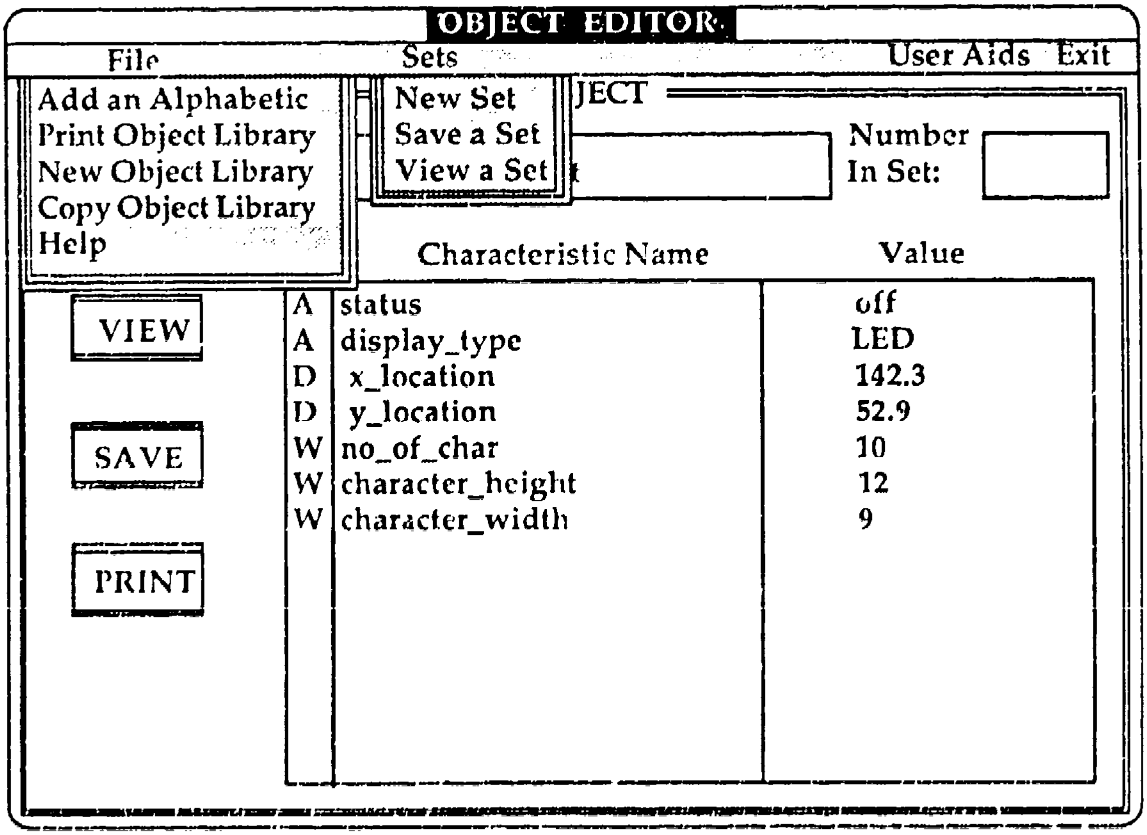
Object Editor Menu Bar

Object Editor contains the following menu options on the menu bar:

- **File** — defines and manipulates the alphabetic dictionary and the object library,
- **Sets** — defines and manipulates object sets,

- **User Aids** — provides the capability to add a value to the dictionary, print the object library, and obtain help messages, and
- **Exit** — terminates the Object Editor and returns the user to the HOSIV screen.

The menu options for the Object Editor are illustrated in the screen below:



	Characteristic Name	Value
A	status	off
A	display_type	LED
D	x_location	142.3
D	y_location	52.9
W	no_of_char	10
W	character_height	12
W	character_width	9

File Commands

The **file** pull-down menu contains commands that allow the user to maintain the alphabetic dictionary and the object library. It contains the following commands:

- **Add an Alphabetic** — generates the dialog window for adding new entries to the alphabetic dictionary.
- **Print Object Library** — allows the user to obtain a printout of all objects in the object library. It first formats the object library and then sends it to the printer.
- **New Object Library** — indicates that a new object library is to be defined and deletes the contents of the current object library.
- **Copy Object Library** — prompts the user to either:
  1. Copy the current object library to the specified library name or
  2. Copy in the specified library to replace the current object library.

Set Commands

The **set** pull-down menu contains commands that allow the user to create, save, and view object sets. It contains the following commands:

- **New set** — defines a new object set. All of the data entry fields in the object dialog window are blanked and the text cursor is placed in the first character in the object name field.
- **Save a set** — saves an object set and creates n objects in the set where n is the number of items in the set. An informative message window is displayed showing the object number being created and the number in the set.
- **View a set**-- creates a list box containing the name of all currently defined sets and permits the user to select a set by pointing

to the desired set name. The user has the option to either:

1. Delete all members in the set and the object set itself by selecting **Delete**,
2. Open the set by selecting **Open**, or
3. **Cancel** the view operation.

For the open set option, the contents of the first member of the set (i.e., object number 1 in the set) is displayed in the object dialog window.

User Aids Commands

The **User Aids** pull-down menu allows the user to obtain additional information about using the Object Editor.

Object Editor Windows

The main Object Editor window is a dialog window for entering object information.

Pushbuttons

The object dialog window, as illustrated above, contains the following pushbuttons:

- **NEW** — clears all input fields and places the text cursor in the object name field.
- **VIEW** — displays a list selection box containing the names of the currently defined objects in the object library (in alphabetic order), including members of set objects. The user can then use the point and click selection method to select an object name. The currently selected object will be highlighted in black. The following pushbuttons are functional in the view window:
  - — **OPEN** — displays the current definition of the selected object in the object dialog window.
  - — **DELETE** — deletes the object currently selected. If the object is a member of a set, a message window is created to inform the user that members of sets cannot be deleted

- **SAVE** — saves the current object as displayed on the screen. The following validation is performed prior to the actual saving of the object definition:

  1. The object and characteristic names are valid names, i.e., they start with an alphabetic character (a-z) and do not contain illegal characters.
  2. The value of a characteristic must be one of the following:
     - Numeric: either whole or decimal number or
     - Alphabetic: the value must be in the alphabetic dictionary.
  3. The characteristic type is determined based upon the entered value. The first letter of the type (W=whole, D=decimal; A=alphabetic) is displayed in the type field of the appropriate characteristic. The default type is WHOLE.
  4. An initial value must be entered for every characteristic name.
  5. For every entered value, a characteristic name was supplied.
  6. If both the characteristic name and value fields are blank, that row of information is ignored.

- **PRINT** — prints the current object definition on the printer.

Text Entry Boxes

The object dialog window contains the following text entry boxes:

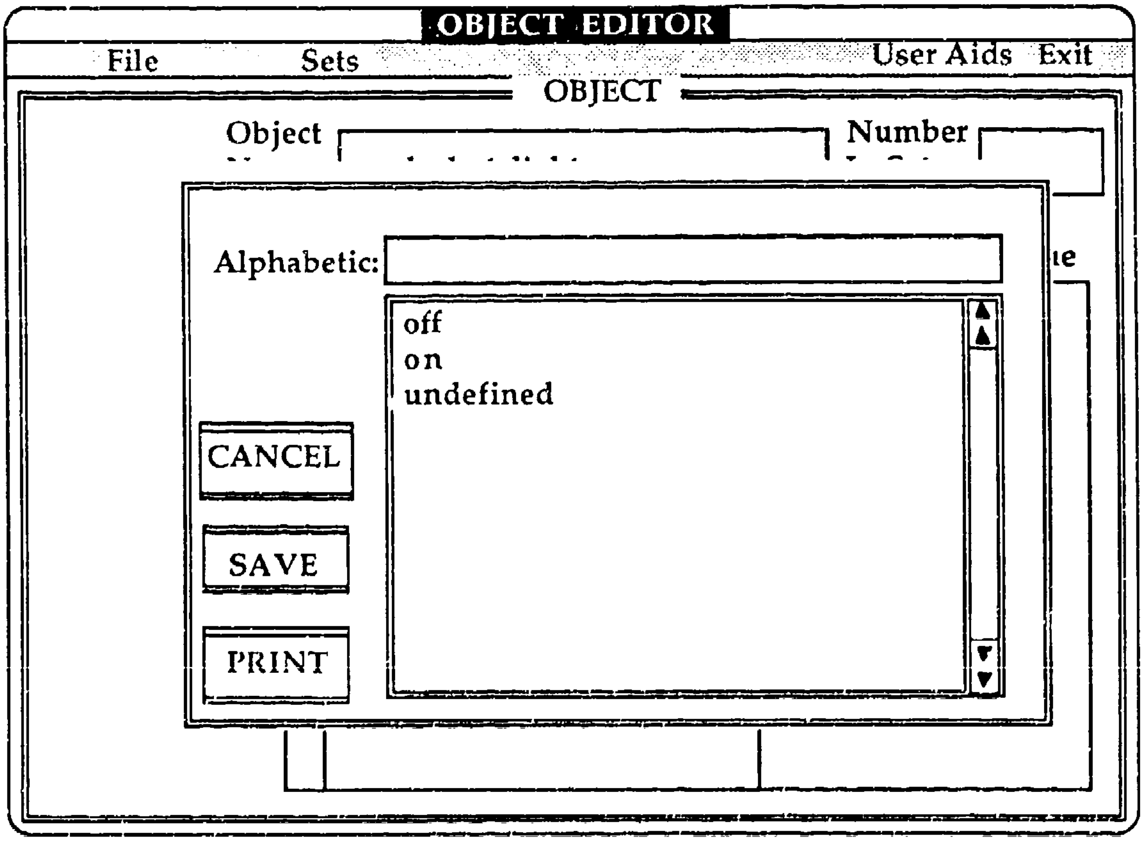
- **Object Name** -- entry of the object name as a maximum of 28 characters. In the example above, the object name is red_alert_light.
- **Number In Set** — entry of the number of members in a set. Used only when the user selects the **Save a Set** option from the **Sets** menu bar. Valid entries are a

number between 2 and 999. In the example above, the object is not a member of a set so this field is empty.

- **Characteristic Name/Value** — entry of a maximum of 14 pairs of characteristic names and values. The type column is automatically filled in by the Object Editor during the SAVE operation. The example above shows the set of characteristics defined for the red_alert_light object. Note that x_location and y_location have been defined as type Decimal since their initial value was a number that included a decimal point. No_of_char, character_height, and character_width are defined as type Whole since their initial value was a number that did not include a decimal point. Status and display_type are type alphabetic since they are not a number. The user should be cautioned to make sure the entered initial value conforms to the type desired. For example, if a characteristic is to be a decimal, then the initial value should be 0.0 not 0.

Adding Alphabetics

Whenever new words are defined, they must be added to the alphabetic dictionary. This is accomplished by selecting the **Add an alphabetic** option from the **File** menu bar. An alphabetic dialog window is displayed that contains a list viewing box that alphabetically lists the currently defined alphabetics in the dictionary.



off
on
undefined

Alphabetic Pushbuttons

The user can select from the following pushbuttons when manipulating alphabetic information:

- **CANCEL** — cancels the add an alphabetic function.
- **SAVE** — adds the entered alphabetic name to the dictionary.

The alphabetic dialog window also contains a text entry box for entry of the alphabetic name.

EVENT EDITOR

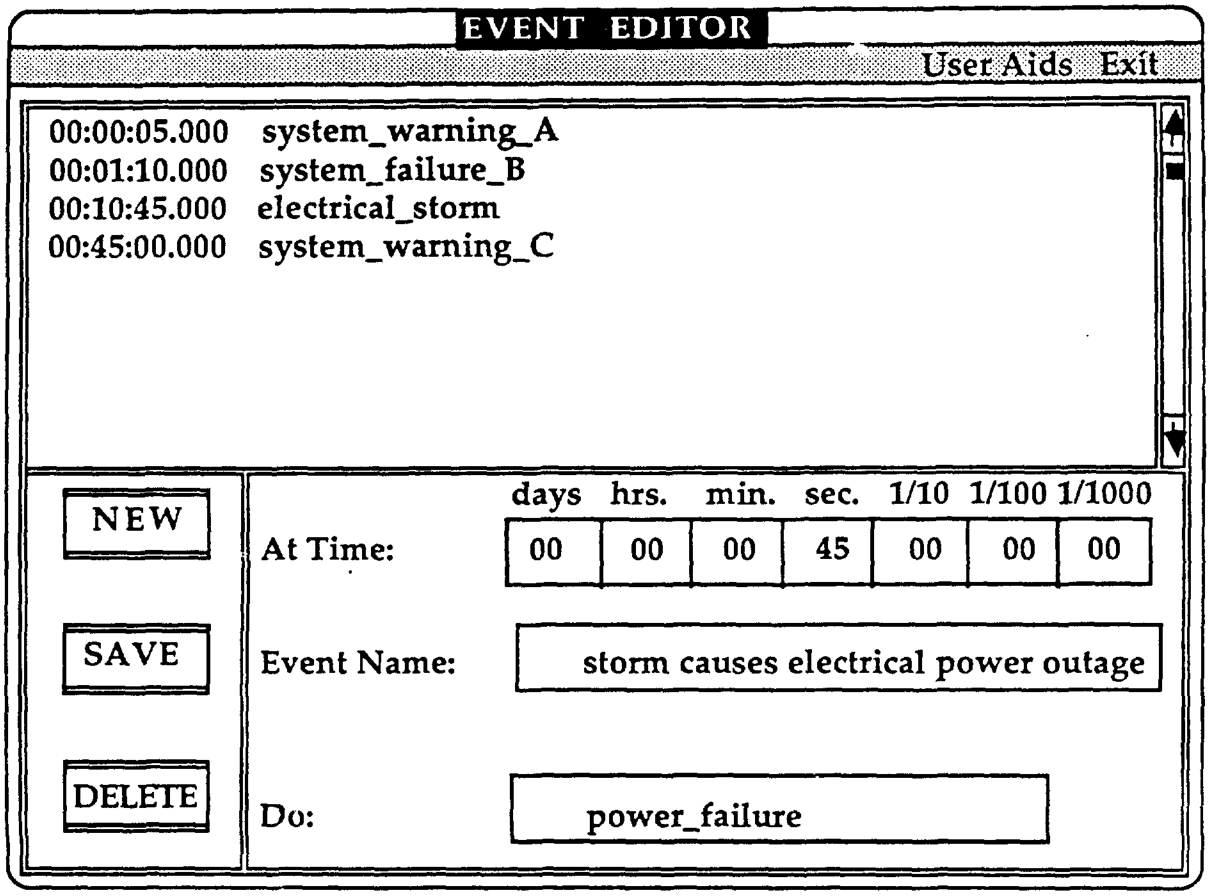
The Event Editor maintains the list of events for the simulation and permits the user to view, edit, delete, and create events. Events are time dependent occurrences which indicate the action that is to be executed at a specific time during the simulation. Events, in conjunction with rules (described in the next section), are the main drivers of what will happen in a simulation. They operate independently of the environment, system, and operator.

At each simulation time, events are always processed prior to evaluating the rule set. That is, events will always occur at the specified time unless the simulation was terminated prior to the event time. Therefore, events can invoke actions that modify objects and/or start or stop rules to impact the course of action. For example, a rule could be defined to process a red alert light if it is lit. The rule can be active throughout the simulation but will never be invoked until the light is lit. An event could be defined to invoke an action that changes the status of the light to turn on at time 150. When the simulation evalutes the rules at time 150, the red alert light rule would be true and the named action would be invoked. Similarly, a group of rules could be defined for evading OPFOR fire. An event could be defined indicating that OPFOR have launched a missile. The action associated with the event could start the rules in the evade fire group.

An event consists of several components: an event description, the name of an action, and the event time at which the event action should be executed. Events typically represent items that represent a system failure at a particular time; a change of environmental conditions such as darkness; or a message from a higher command. Events can also be used to evaluate alternative workloads such as doubling the items to be processed during a certain time period. It is useful to develop the set of events early in the simulation process to generate an overall time line of activities.

Event Editor Screens

The Event Editor screen is illustrated below:



```
00:00:05.000   system_warning_A
00:01:10.000   system_failure_B
00:10:45.000   electrical_storm
00:45:00.000   system_warning_C
```

	days	hrs.	min.	sec.	1/10	1/100	1/1000
At Time:	00	00	00	45	00	00	00

Event Name: storm causes electrical power outage

Do: power_failure

Title Bar

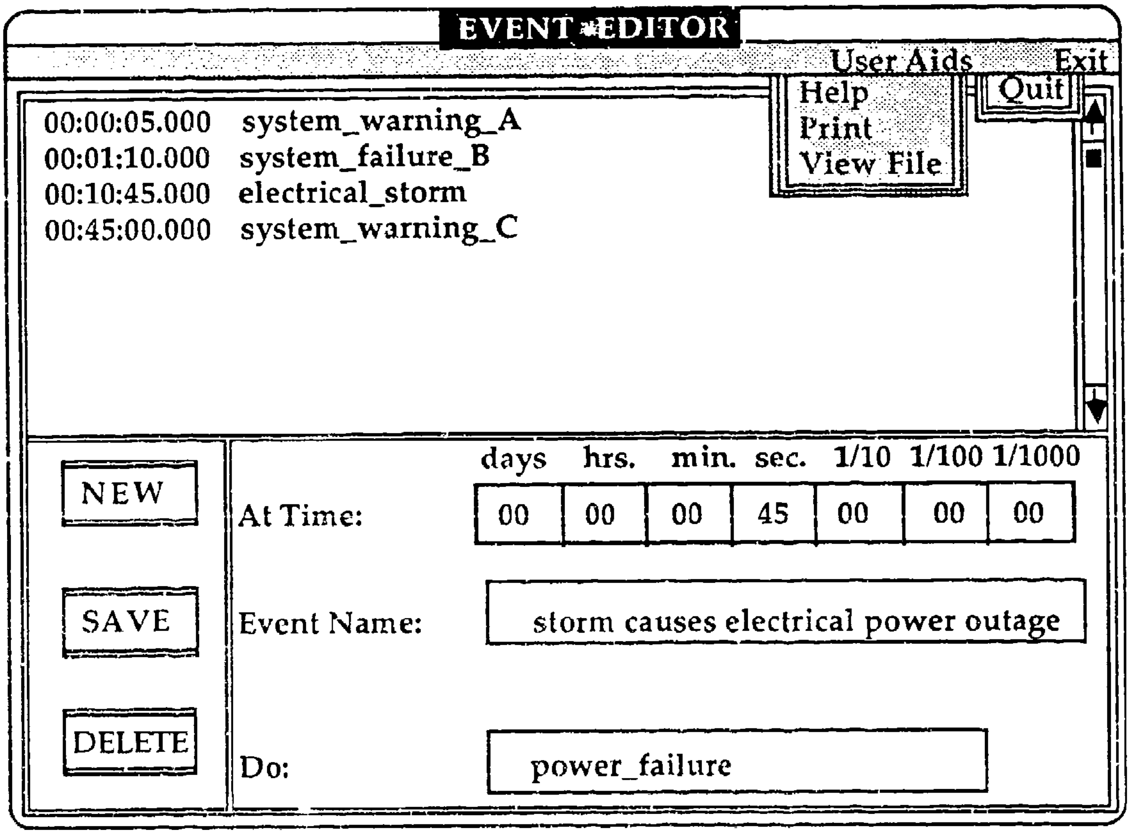
The title bar of Event Editor contains the words 'Event Editor' as the function name.

Event Editor Menu Bar

Event Editor contains the following menu options on the menu bar:

- **User Aids** — provides the capability to print the list of defined events, view the contents of any action file, and obtain help messages, and

- **Exit** — terminates the Event Editor and returns the user to the main HOS-IV screen.



00:00:05.000 system_warning_A
00:01:10.000 system_failure_B
00:10:45.000 electrical_storm
00:45:00.000 system_warning_C

	days	hrs.	min.	sec.	1/10	1/100	1/1000
At Time:	00	00	00	45	00	00	00

Event Name: storm causes electrical power outage

Do: power_failure

User Aids Commands

The **User Aids** pull-down menu contains commands that allow the user to view other files, print all currently defined events, and obtain help messages. It contains the following commands:

- **View Files** — allows the user to obtain a window that displays currently defined actions.
- **Print Actions** — allows the user to obtain a listing of all defined actions.

- **Help** — allows the user to obtain additional information about using the Event Editor.

Event Editor Windows

The main Event Editor window is a dialog window for entering event information.

Pushbuttons

The event dialog window contains the followi.ng pushbuttons:

- **NEW** — clears all input fields and places the text cursor in the event name field.
- **DELETE** — deletes the currently selected event.
- **SAVE** — saves the current event as displayed on the screen. The following validation is performed prior to the actual saving of the event definition:

  1. The Event time is compared to the end simulation time. If the Event time is greater than the end simulation time, an error message window is displayed indicating that the Event time is too large and the user must enter the correct information.
  2. The Do action is evaluated to determine if the action name is valid and has been defined. If it has not, a warning message window is displayed indicating that the action is undefined. The action name should be reviewed to determine if it was entered correctly or not. If the action name is correct, then the user must remember to subsequently define the action using the Action Editor.

Text Entry Boxes

The event dialog window contains the following text entry boxes:

- **At Time** — entry of the time in the form of seven two digit numbers, one for each time unit. In the sample above, the time is 45 seconds.
- **Event Name** — entry of the Event name as a maximum of 28 characters. In the example above, the event name is 'storm causes electrical power outage'.
- **Do** — entry of an action name. In the example above, the action name is 'power_failure'.

List Selection Box

A list selection box showing the event time, description, and event action of currently defined events is displayed in the upper left corner of the event dialog window as shown above. The set of events included in the illustration above includes: system_warning_A, system_failure_B, electrical_storm, and system_warning_C.

RULE EDITOR

The Rule Editor maintains the list of defined rules for a simulation and permits the user to view, edit, delete, and create rules. Rules set up conditions that determine the action to be taken depending on specified conditions. A rule is defined by the following parameters: 1) a starting condition, 2) an ending condition, 3) the name of the action to be invoked if the starting condition is true, and 4) a unique rule number. Rules are grouped according to type and three distinct sets of rules are included for each simulation: Hardware, Operator, and Environment.

Rule Definition

A rule consists of five elements: 1) a rule number; 2) a rule name; 3) an if clause; 4) a do clause; and 5) an until clause as described below.

Rule Number

The rule number is a unique number. For hardware and environment rules, it is a two digit number 00 (lowest) to 99(highest). For operator rules, it is a unique three digit number consisting of a one digit priority assignment from 0 (lowest) to 9 (highest) and a two digit number 00 (lowest) to 99 (highest). For example, rules 204, 210, 225, 267, and 298 would all be in group 2 and rule 298 would be evaluated prior to rule 267.

Rule Name

The rule name is an alphanumeric name containing a maximum of 28 characters. The characters can be any letter a-z, digit (0-9), and underscore (_). The first character must be a letter.

If Clause

The If clause indicates starting condition for when the rule is to be invoked. It is expressed as <item> relationship <item> where:

<item> is a number, alphabetic, or characteristic of an object and

relationship is a boolean operator including EQUALS, GREATER_OR_EQUAL, GREATER_THAN, LESS_OR EQUAL, LESS_THAN, or NOT_EQUAL_TO.

Examples of If clauses include:

number_of_objects GREATER_THAN 0,

color OF object EQUALS red,

size OF object1 EQUALS size OF object2, and

sim_time NOT_EQUAL_TO 0.

Do Clause

Do clauses contain the name of the action to be invoked if the rule evaluation conditions are met.

Until Clause

The Until clause contains the condition that indicates when the rule is to be terminated. It is expressed as <item> relationship <item> where:

<item> is a number, alphabetic, or characteristic of an object and

relationship is a boolean operator including EQUALS, GREATER_OR_EQUAL, GREATER_THAN, LESS_OR EQUAL, LESS_THAN, or NOT_EQUAL_TO.

Examples of Until clauses include:

number_of_objects EQUALS 0,

color OF object EQUALS green, and

status OF object1 EQUALS status OF object2.

Active Rules

Rules are evaluated by the simulation at the beginning of each simulation time interval. Only active rules are evaluated. Rules are activated (i.e., put on the list of rules that should be evaluated during the time interval) and suspended (i.e., removed from the list of rules that should be evaluated during the time interval) by actions. The HOS Action Language (described in Section 7) contains a set of verbs for activating all rules (hardware, operator, and environment), a group of rules (operator rules only), or a specific rule (hardware, operator, and environment). Actions can also suspend rules using similar verbs to prevent rules from being evaluated.

Rule Priorities

Each rule is assigned a unique number that indicates its priority with 0 representing the lowest and 999 the highest. The set of active rules are evaluated during a time interval according to each rule's priority. The action associated with a high priority rule may contain statements to suspend lower priority rules If operator rules are grouped by mission phases, a critical phase may contain rules numbered 700 to 799. Rule 799 could activate an action that suspends all rules in groups 2 through 6. This results in rules being removed from the active rule list which will not be evaluated until the rules are started by an action.

Evaluating Rules

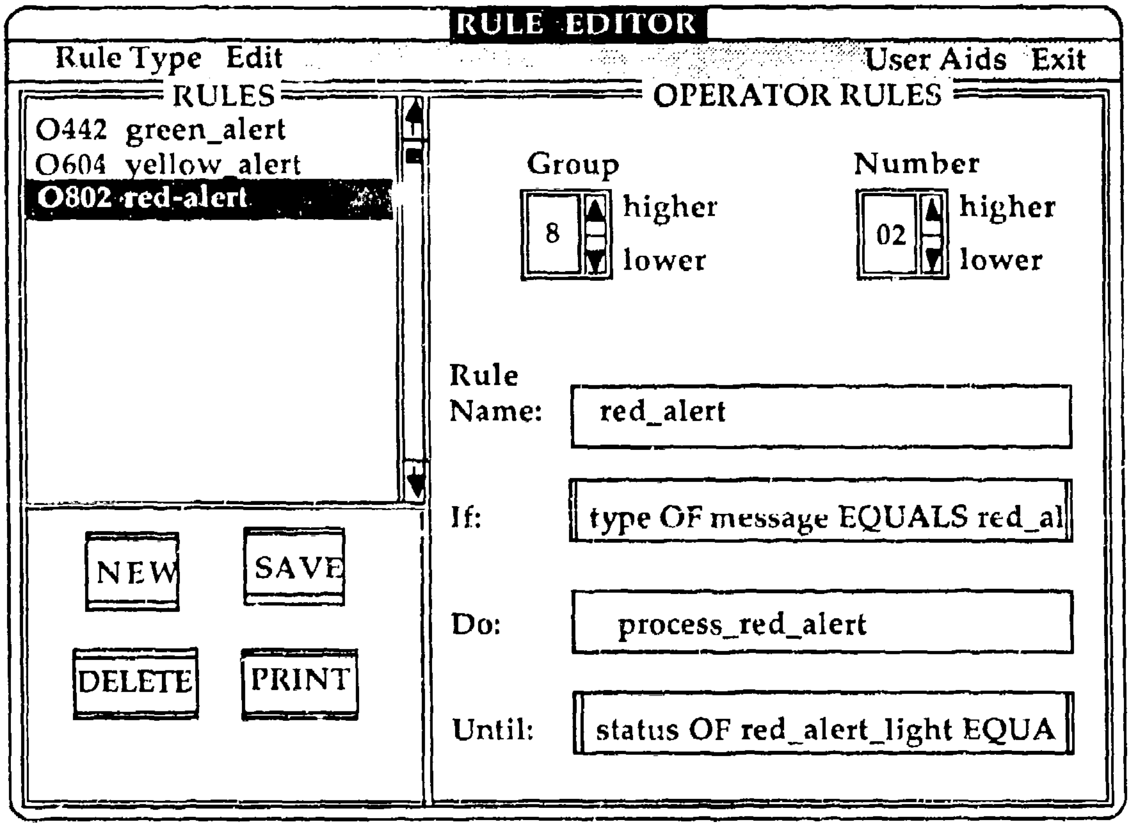
If a rule is on the active list, the ending condition is evaluated first. If false, then the starting condition is evaluated. If the starting condition is true, the named action will be invoked. At each simulation time interval, the rule will be reevaluated and only if **both** conditions are met will the action occur. It a rule is to occur throughout the simulation, a rule such as shown below should be defined:

If:	sim_time GREATER_OR_EQUAL 0
Do:	continuous_action *(action name)*
Until:	status OF simulation EQUALS done

The conditions for activating a rule would be met throughout the entire simulation as long as the rule was on the active rule list, i.e., it had been started by and not suspended by an action.

Rule Editor Screens

The Rule Editor screen is illustrated below:



O442 green_alert
O604 yellow_alert
O802 red-alert

Group: 8

Number: 02

Rule Name: red_alert

If: type OF message EQUALS red_al

Do: process_red_alert

Until: status OF red_alert_light EQUA

Title Bar

The title bar of Rule Editor contains the words 'RULE EDITOR' as the function name. Rule Editcr does not use the current activity or status information areas of the title bar.

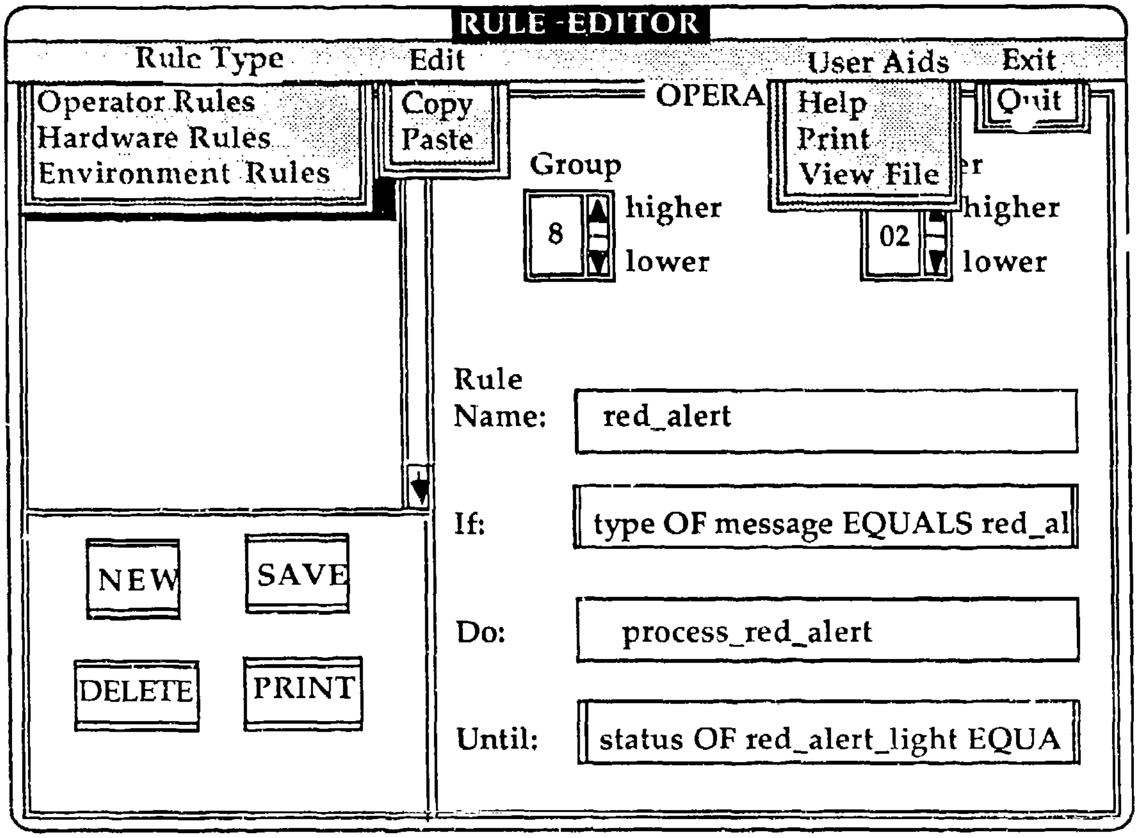
Rule Editor Menu Bar

Rule Editor contains the following menu options on the menu bar as illustrated below:

- **Rule Types** — defines the rule type as environment, hardware, or operator. When one option is selected, the current rules are saved and the rules associated with the selected type are loaded.
- **Edit** — provides copy and paste functions. **Copy** copies the current rule into an invisible clipboard buffer. **Paste** copies the clipboard buffer into the rule currently defined on the screen only.
- **User Aids** — provides the capability to print the list of defined rules, view the contents of any action file, obtain help, and
- **Exit** — terminates the Object Editor and returns the user to the main HOS-IV screen.

Rule Editor Pull-Down Menus

The Rule Editor pull-down menu contains commands that allow the user to define the rule type as illustrated below.



Rule Name: red_alert

If: type OF message EQUALS red_al

Do: process_red_alert

Until: status OF red_alert_light EQUA

Rule Type Commands

The Rule Type pull-down menu contains the following commands:

- **Environment** — defines an environment type. All of the data entry fields in the rule dialog window are blank and the text cursor is placed in the first character position in the rule name field.

- **Hardware** — defines a hardware type. All of the data entry fields in the rule dialog window are blank and the text cursor is placed in the first character position in the rule name field.
- **Operator** — defines an operator type rule. All of the data entry fields in the rule dialog window are blank and the text cursor is placed in the first character position in the rule name field.

User Aids Commands

The **User Aids** pull-down menu contains commands that allow the user to view other files, print all currently defined tasks, and obtain help. It contains the following commands:

- **View Files** — allows the user to obtain a window that displays currently defined actions.
- **Print rules** — allows the user to obtain a printout of all defined rules. Rules are formated and then printed on the line printer.
- **Help** — allows the user to obtain additional information about using the Rule Editor.

Rule Editor Windows

The main Rule Editor window is a dialog window for entering rule information.

Pushbuttons

The rule dialog window (see previous illustration) contains the following pushbuttons:

- **NEW** — clears all input fields and places the text cursor in the rule name field.
- **DELETE** — deletes the currently selected rule.

- **SAVE** — saves the current rule as displayed on the screen. The following validation is performed prior to the actual saving of the rule definition:

  1. The rule name is a valid name, i.e., it starts with an alphabetic character (a-z), does not contain any illegal characters, and is unique.
  2. The If and Until clauses contain valid boolean conditions. If the boolean operator or constant is invalid, an error message window is displayed indicating that the clause cannot be saved. If a characteristic-object pair is undefined, a warning message window is displayed indicating that there is a problem with the boolean conditions.
  3. The Do action is evaluated to determine if the action name is valid and has been defined. If it has not, a warning message window is displayed indicating that the action is undefined.

- **PRINT** — prints the current rule definition on the printer.

Text Entry Boxes

The rule dialog window contains the following text entry boxes:

- **Rule Name** — entry of the rule name as a maximum of 28 characters. In the screen above, the rule name is red_alert.
- **If** — entry of the if clause in the form:

  <item> relationship <item>

  where item can be a characteristic of an object, an alphabetic, or a number and relationship can be equals, not_equal, less_than, less_or_equal, greater_than, or greater_or_equal. In the example above, the if clause is 'type OF message EQUALS red_alert'. 'type OF message' is an item representing a characteristic of an object;

EQUALS is the boolean operator, and 'red_alert' is an alphabetic.

- **Do** — entry of an action name. In the example above, the action name is process_red_alert.
- **Until** — entry of the until clause in the form:

  <item> relationship <item>

  where value can be a characteristic of an object, an alphabetic, or a number and relationship can be equals, not_equal, less_than, less_or_equal, greater_than, or greater_or_equal. In the example above, the until clause is status OF red_alert_light EQUALS processed. Since the until clause box can contain only 28 characters, only the first 28 characters of the clause is displayed. The remainder of the rule can be displayed by using the mouse to move the mouse cursor to the last character and using the arrow keys to shift the contents of the box right and/or left.

Click Boxes

The rule dialog window contains a click box for entry of the rule group number 0-9 (operator rules only) and rule number 0-99 (environment and hardware rules). A list selection box showing the number and names of currently defined rules is displayed in the upper left corner of the rule dialog window as shown above. To change one of the numbers, use the mouse to move the mouse cursor to the higher or lower arrow and click the mouse once for each change in the value of the current contents.

Rule Editor Error Messages

When the user completes entering a rule and depresses the save pushbutton to save the rule, the entered fields are validated. Error messages indicate errors that must be corrected before the rule can be saved; warning messages are informative messages indicating that something suspect is contained in the rule but it does not have to be corrected before the rule can be saved. The following error messages are generated for If and Until statements:

1. A piece of the statement is missing,
2. Syntax error: (specific problem),
3. Undefined alphabetic or syntax error,
4. Syntax error, and
5. A rule with that number already exists.

Warning messages appear when undefined entities (object, characteristic, or action) are used in the rule definition. The window indicates where the problem is, what is undefined, and what name the user entered.

ACTION EDITOR

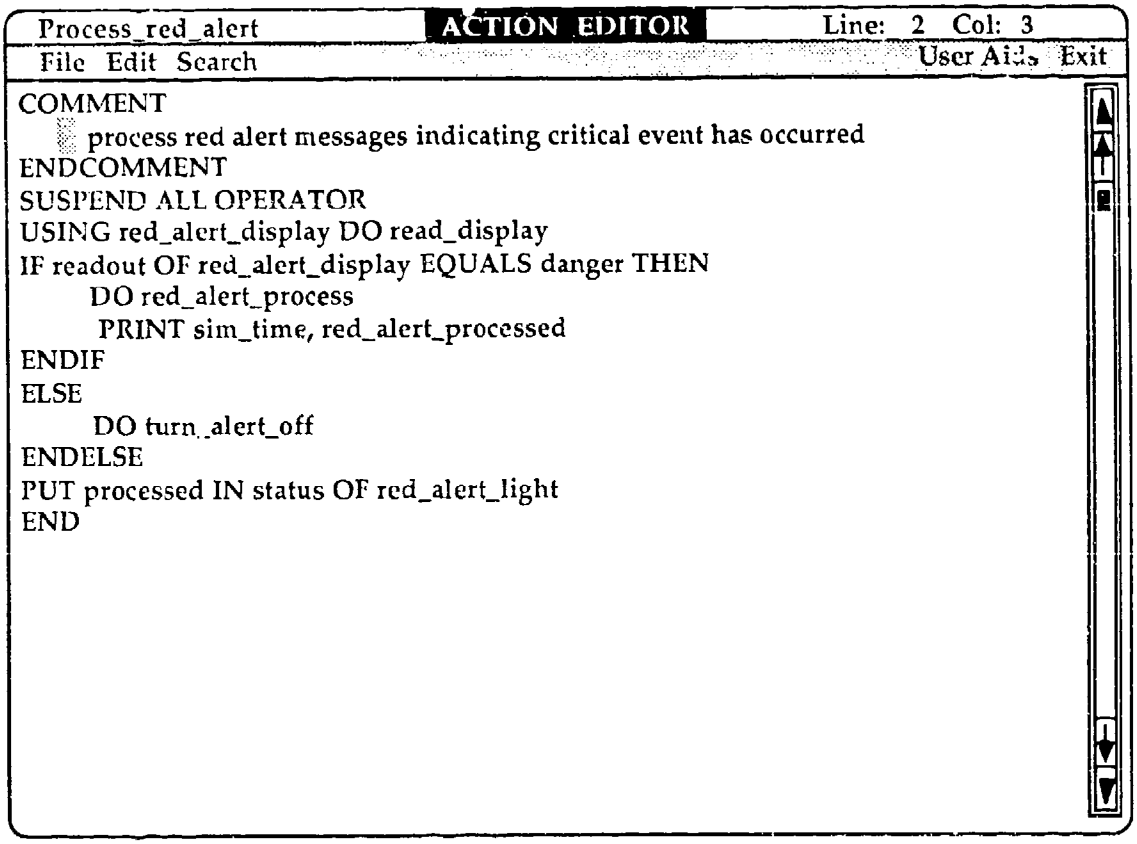
The Action Editor is used to enter actions. Actions describe the steps required to accomplish a process by the operator, system, or environment. Actions can include updates to the values of object characteristics, invocation of other actions, and the initiation or suspension of action rules.

Actions are defined using a small set of standard verbs (e.g., PERFORM, SET, SUSPEND) known as the HOS Action Language (HAL). A description of the verbs is presented in Section 7.

Action Editor is essentially a free format word processor with word wrapping, cut and paste features, and mouse and keypad control of the text cursor. Once an action is entered, the user can specify that it be translated by HAL for use in the simulation. The status of the translation, whether successful or errors were detected, is displayed in a message window. If any errors were detected in the translator, the View File option can be used to obtain a separate window containing the translator output and accompanying error message.

Action Editor Screens

The Action Editor screen consists of a title bar, a menu bar, a text editing window with a scrollbar, and a number of dialog boxes used for program interaction with the user and is illustrated below:



```
COMMENT
   process red alert messages indicating critical event has occurred
ENDCOMMENT
SUSPEND ALL OPERATOR
USING red_alert_display DO read_display
IF readout OF red_alert_display EQUALS danger THEN
       DO red_alert_process
        PRINT sim_time, red_alert_processed
ENDIF
ELSE
       DO turn_alert_off
ENDELSE
PUT processed IN status OF red_alert_light
END
```

Action Editor Title Bar

The Action Editor title bar consists of a current activity area on the left which displays the name of the current action, the words 'ACTION EDITOR' in the center, and the current line and column position of the text cursor on the right. In the example above, the current action is named 'process_red_alert'. The text cursor is currently located on line 2 in column 3.

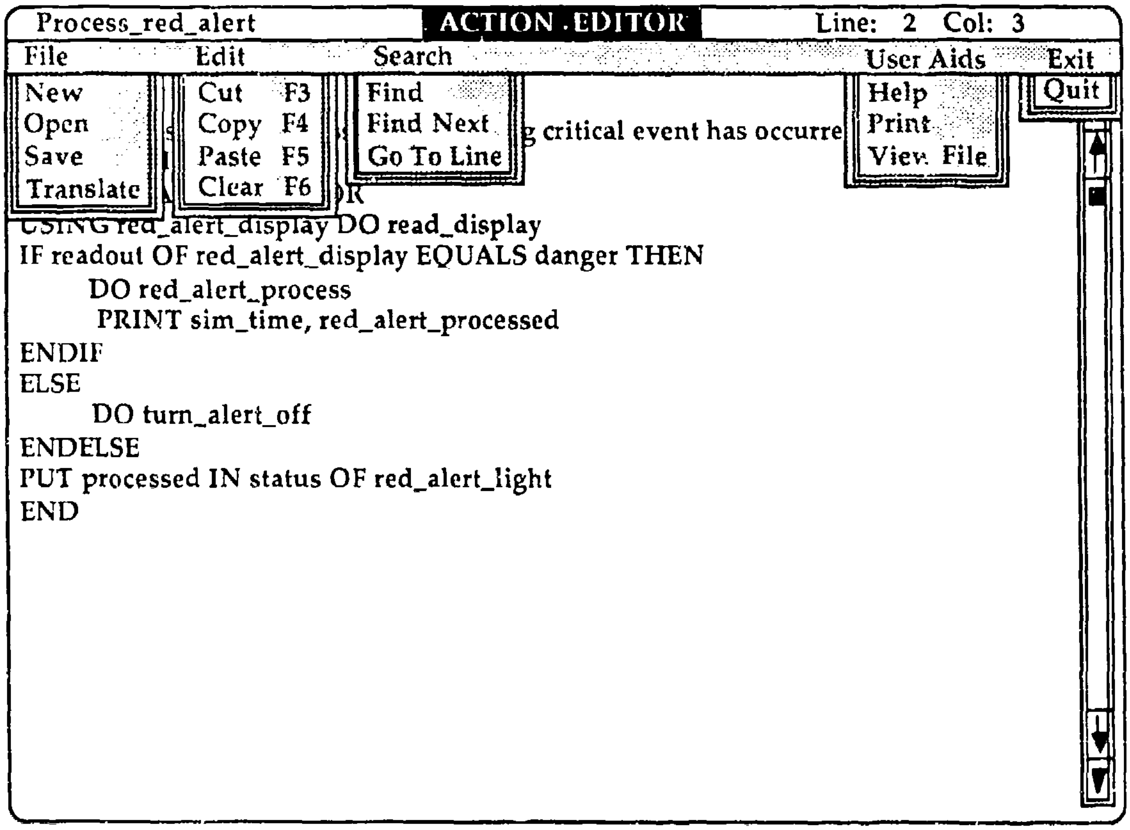
Action Editor Menu Bar

The menu bar for the Action Editor contains the following menu options:

- **File** — tile related commands such as saving, opening, etc.
- **Edit** — editing commands: cut, paste, etc.
- **Search** — word and line search commands.
- **User Aids** — provides the capabilities to view help files and action files.
- **Exit** — terminates Action Editor and returns the user to the HOS-IV module.

Pull-Down Menus

The pull-down menus for the Action Editor are illustrated below.



```
g critical event has occurre
)R
USING red_alert_display DO read_display
IF readout OF red_alert_display EQUALS danger THEN
        DO red_alert_process
         PRINT sim_time, red_alert_processed
ENDIF
ELSE
        DO turn_alert_off
ENDELSE
PUT processed IN status OF red_alert_light
END
```

File Commands

The **File** pull down menu contains:

- **New** — closes the current document after asking if changes should be saved and then creates a new empty action.

- **Open** — closes the current document after asking if changes should be saved and then opens the file selection dialog box.
- **Translate** — asks if changes to the current document should be saved and then invokes the HAL translator.
- **Save** — opens the save dialog box to request the action name for the action to be saved.

Edit Commands

The **Edit** pull down menu contains the commands listed below. The Edit functions are also available as function keys as indicated in parenthesis.

- **Cut** — removes selected text from the current document and stores it in the clipboard (F3).
- **Copy** — copies selected text from the current document and stores it in the clipboard (F4).
- **Paste** — inserts the contents of the clipboard in the current document at the insertion point (F5).
- **Clear** — deletes the selected text from the current document and throws it away (F6).

Search Commands

The **Search** pull down menu contains:

- **Find** — opens the input search string dialog box for the user to define a search string.
- **Find Next** — finds the next occurrence of the string defined in the input search string dialog box.
- **Goto Line** — opens the input line number dialog box for the user to enter the number of the line to goto.

User Aids Commands

The **User Aids** pull down menu contains commands that allow the user to receive help on the current module and view action files:

- **Help** — allows the user to obtain additional information about using ACTION EDITOR.
- **Print** — allows the user to get a formatted hardcopy on the standard printer.
- **View File** — allows the user to view action files created using the Action Editor.

Action Editor Text Editing Window

The text editing window consists of a text area and a scrollbar. The text area is 23 rows by 77 columns. The scrollbar is to the right of the text area and has four control buttons and a relative file position indicator (■). The relative file position indicator shows the current location of the text cursor relative to the length of the file, e.g. if the indicator is located in the middle of the box, then the text cursor is located in the middle of the file. The four controls on the scrollbar are:

- **Scroll Line Down** (▲)— displays a page of text starting from the line before the current top line.
- **Scroll Page Down** (↑)— displays a page of text starting one page before the current top line.
- **Scroll Page Up** (↓)— displays a page of text starting from the current bottom line.
- **Scroll Line Up** (▼)— displays a page of text starting from the line after the current top line.

Text Entry Fields

The dialog boxes used by the Action Editor contain text entry fields in addition to pushbuttons, list boxes, and other message window components.

- **File name** — user can edit the current action name and choose to save or cancel the operation.
- **Search string** — user can input a string of characters to search for. The search string can contain only alphanumeric characters and underscore (_). It cannot search for control characters such as tabs.
- **Line number** — user can enter an integer line number only and choose to move the text cursor to that line or use the goto option.

Translating Actions

Once an action has been entered, it must be translated by HAL in order to be included in the simulation. To translate an action, the **Translate** option on the **File** pull-down menu is selected. While the action is being translated, an informative screen is displayed. HAL, the HOS Action Language, processes each word in the action and when it detects an error, a message window is displayed indicating that HAL found an error. The action code and the results of the HAL translation, including appropriate syntax error messages, are contained in the TRANSLATOR OUTPUT file. This file can be view by selecting the VIEW FILE command from the User Aids pull-down menu. Another useful file is the SYMBOL TABLE file which lists the name and type of variables used by the action currently being translated. This file is accessible to the user from the View File option of the User Aids pull-down menu and is named SYMBOL TABLE.

These files also can be accessed by opening the file directly through the file menu in the Action Editor.

Action Editor Message Windows

The Action Editor message windows include the following:

- **String not found** — the search string was not found in the document.
- **Translation successful** — the action was successfully translated.
- **Translation unsuccessful** — the action did not translate.
- **End editing session** — the user may confirm or cancel his Quit selection from the Exit menu.
- **Save changes** — the user may confirm or cancel the saving of the current action.
- **Out of memory** — the document is too large (actually if the file is this big the compiler won't accept it anyway).
- **File selection** — the user can select an action to open, delete, or cancel the selection.
- **Confirm delete** — the user can confirm or cancel the decision to delete an action.
- **Can't delete current file**— the user attempted to delete the currently opened file.
- **Can't delete system file** — the user attempted to delete the TRANSLATOR OUTPUT file or the SYMBOL TABLE file.

Action Editor Information Windows

The Action Editor information messages are listed below:

- **Printing in progress** — the printing of an action is underway.
- **Reading file** — the file is currently being read by the system as a result of a user initiated action.

- **Translating** — the HAL translator is running.

Action Editor Function Keys

Action Editor uses the following function keys for text editing capabilities:

- **F1** - Begin Mark — marks the beginning of text selection area. The mouse is used to move the cursor to the point where the text begins.
- **F2** - End Mark — marks the end of text selection area. The mouse is used to move the mouse cursor to the point where the text ends.
- **F3** - Cut — the selected text area is deleted from the text and put into a temporary storage area that can be used for paste operation.
- **F4** - Copy — the selected text area is copied into a temporary storage area that can be used for paste operation.
- **F5** - Paste — the contents of the temporary storage area is copied into the action at the current text cursor location.
- **F6** - Clear — the selected text area is deleted from the text but the contents of the temporary storage area are not affected.

7. HOS ACTION LANGUAGE

HOS provides a special purpose language, the HOS Action Language (HAL), to write actions that specify the series of steps required to accomplish a simulation. This section describes HAL and the associated verbs.

Actions represent distinct modular processes. For example, the process_yellow_alert light woJld contain the steps the operator would follow in order to respond to a yellow alert light. The maximum number of statements in a single action (excluding comments) is 150 statements.

HAL TERMS

The following terms are used within this discussion of HAL.

Verbs

Verbs refer to the set of distinct HAL commands that are used to specify actions. Each HAL statement must begin with a verb. Each verb must be in the first character position of a statement.

Statement

A HAL statement begins with a verb and can contain up to 256 characters. No statement terminator is required.

Constant

A constant is a whole number or decimal that is used in an action statement to specify a parameter that is to contain a numerical value, for example 25 or 125.34.

Expression

An expression is used to specify a condition such as A equals B. It consists of two parameters joined by a logical operator in the following form:

<item> logical operator <item>

Parameters must be a constant, alphabetic, variable, or characteristic of an object. The logical operators include:

```
EQUALS
NOT_EQUAL_TO
GREATER_OR_EQUAL
LESS_OR_EQUAL
GREATER_THAN
LESS_THAN
```

List

A list is a group of parameters that can be contained in an action statement. The parameters can be separated by commas or spaces. An example of a list of whole number variables is:

```
WHOLE whole_number_1, whole_number_2,
    whole_number_3
```

Text

Text refers to any set of letters or words and can be used to specify English-like sentences for comments and output. Text can contain any characters and numbers except single and double quotes.

Variable

A variable is declared within an action to contain an alphabetic, decimal number, or whole number value. An example of a variable for a HOS simulation would be action_time which could be defined as a decimal number to contain the amount of time required for an action. The variable name must conform to the HOS naming conventions and contain 1 to 28 alphanumeric characters (i.e., a-z and 0-9) with the first letter an alphabetic (i.e., a-z). The name cannot contain any embedded spaces or punctuation except for an underscore (_). Valid variable names are shown below:

```
a_whole_number
distance_123
anumber
```

Invalid variable names are shown below:

a-whole-number	Contains an invalid character — dash
123name	First letter is not a letter (a-z)
a number	Contains an invalid character _ space

If the variable is to be used within a calculation, it must be either a decimal or whole number. Variable types are specified using the DEFINITIONS verb in the beginning of the action as described in more detail in a subsequent section.

Verb Blocks

Several HAL verbs permit groups of statements to be specified in blocks that begin with the verb name and are terminated with END verb name. For example, when specifying comments, all comment text are enclosed between COMMENT and ENDCOMMENT statements.

Verb templates are provided for some of the verbs to illustrate the verb syntax. The gray boxes for these templates contain HOS verbs.

ORGANIZATION

The remainder of this section describes in detail the HOS verbs. For some verbs, verb templates are provided to illustrate the verb syntax. In this template, the gray boxes contain HOS verbs. In the examples contained in this section, all HOS verbs and expressions are shown in upper case.

The HOS verbs and the descriptions in this section are grouped into the following functional groups:

- Informational — COMMENT, DEFINITIONS, FILE, and PRINT;

- Action Manipulation — USING...DO and RECEIVE;
- Rule Manipulation — START and SUSPEND;
- Object Manipulation — GET, PUT, and RETRIEVE;
- Computational — SET;
- Conditional — IF and WHILE; and
- Terminating — END and END_SIM.

In order to look up a verb directly, the following contains the verbs listed in alphabetical order:

Comment/Endcomment	7-5
Definitions/Enddefinitions	7-6
Else/Endelse	7-30
End	7-41
End_Sim	7-42
Exitwhile	7-33
File/Endfile	7-10
Get	7-19
If/Endif	7-30
Print/Endprint	7-13
Put	7-21
Receive/Endreceive	7-38
Retrieve	7-23
Set	7-26
Start	7-15
Start_C_Code/End_C_Code	7-40
Suspend	7-17
Using/Do	7-36
While/Endwhile	7-33

INFORMATIONAL VERBS

Informational verbs are used to provide information about the simulation both within an action and to external sources such as the printer and simulation output files. Informational verbs include COMMENT, DEFINITIONS, FILE, and PRINT.

The COMMENT/ ENDCOMMENT Verb

The COMMENT verb is used for the insertion of any textual material by the user at any point in the action. Each COMMENT must be paired with a terminating ENDCOMMENT. The ENDCOMMENT does not have to be on the same line as the COMMENT verb.

Syntax

The COMMENT verb syntax is illustrated below. The user's textual material is enclosed between the COMMENT and the ENDCOMMENT statements.

COMMENT
text
ENDCOMMENT

where:

<text> contains any combination of alphanumeric characters and special symbols except single and double quotes

Examples

Examples of using the COMMENT/ ENDCOMMENT verb are shown below:

Example 1

```
COMMENT
   Action A sets up the parameters to move the
operator's right-hand from the current-location to
the desired-location using Fitt's law
ENDCOMMENT
```

Example 2

```
COMMENT Standard eye-movement algorithm
ENDCOMMENT
```

Example 3

```
COMMENT Modified algorithm ENDCOMMENT
```

The DEFINITIONS/ ENDDEFINITIONS Verb

The DEFINITIONS verb is used to categorize variables within a single action as one of the following types — alphabetic, characteristic, decimal variable, local object, object, set object, or whole variable. The definitions must occur at the beginning of the action and can only be preceded by comment statements. Only the variable types actually used in an action need be included in the definitions. More than one variable name can be defined in a single definition statement. The multiple variable names can be separated by commas or spaces. Each DEFINITIONS statement must be paired with a terminating ENDDEFINITIONS statement.

The variable types include the following:

ALPHABETIC — alphabetic values (or words) that have previously been defined and included in the alphabetic dictionary.

CHARACTERISTIC — the name of the object characteristic. If multiple objects have the same characteristic, the characteristic name need only be defined once. For example, if both old_emitter and new_emitter objects have a characteristic, named status; status need only be defined once in the list of characteristics.

DECIMAL — decimal number values.

LOC_OBJECT — the name of an object that will be used to store object information for a member of an object set.

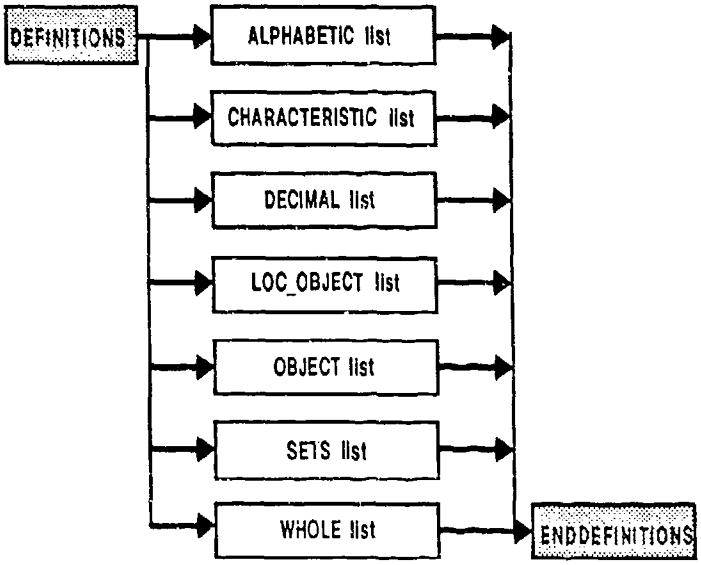
OBJECT — object names of singularly declared objects.

SETS — names of object sets.

WHOLE — whole number values.

Syntax

The syntax of the DEFINITIONS/ ENDDEFINITIONS verb is shown below. The user's variable definitions are enclosed between the DEFINITIONS and the ENDDEFINITIONS statements.



where:

<def-statement> contains any one of the following:

ALPHABETIC	[variable list]
CHARACTERISTIC	[characteristic list]
DECIMAL	[variable list]
LOC_OBJECT	[object list]
OBJECT	[object list]
SETS	[set objects list]
WHOLE	[variable list]

Examples

Examples of using the DEFINITION/ ENDDEFINITIONS verb are shown below:

Example 1

The example below illustrates the use of the definition verb to define the variables included in an action. Within the action, the variables are typed as folows;

- screen_size and move_time are declared as whole numbers.
- x_location, y_location, z_location, distance, and move_time are declared to be decimal numbers.
- x_position, y_position, z_position, x_eye, y_eye, z_eye, x_hand, y_hand, and z_hand are declared to be characteristics of objects. Both the eye and hand objects have characteristics named x_position, y_position, and z_position.
- eye and hand are declared to be objects.

```
DEFINITIONS
   WHOLE screen_size, move_time
   DECIMAL x_location, y_location, z_location
   DECIMAL distance, move_time
   CHARACTERISTIC x_position, y_position,
         z_position, x_eye, y_eye, z_eye,
         x_hand,y_hand, z_hand
   OBJECT eye, hand
ENDDEFINITIONS
```

Example 2

This example illustrates an alternative way to specify the variables for whole and decimal types to improve readability.

```
DEFINITIONS
   WHOLE
               screen_size,
               move_time
   DECIMAL
               x_location,
               y_location,
            z_location
ENDDEFINITIONS
```

Example 3

The definitions below illustrate individual members of an object set within an action. The object set name would be specified as SETS; the name of the individual member would be specified as LOC_OBJECT; and the object characteristics would be specified as CHARACTERISTICS.

```
DEFINITIONS
    LOC_OBJECT action_object_name
    SETS object_set_name
    CHARACTERISTIC object_charcteristic
ENDDEFINITIONS
```

This example illustrates the use of a local object named an_emitter associated with an object set named emitters. The characteristics include frequency, time_on, and time_off.

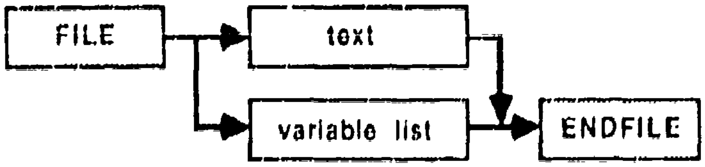
```
DEFINITIONS
    LOC_OBJECT an_emitter
    SETS emitters
    CHARACTERISTIC frequency, time_on,
            time_off
ENDDEFINITIONS
```

The FILE/ENDFILE Verb

The FILE verb is used to store values in a file for later printing or processing. Each FILE statement creates a single record in the file. The records created by use of the FILE verb will be entered into the file as they occur during the simulation. The user can include an explanatory text string. The items included within the FILE statement may be separated by commas or spaces and can occur in any order. Each FILE statement must be paired with a terminating ENDFILE. The resulting file is accessible from the View Results output (see Section 10) and is stored in the file named USER SIMULATION OUTPUT.

Syntax

The syntax of the FILE/ENDFILE verb is shown below:



where:

<text> contains any series of alphanumeric characters that is not a name of a HOS item, e.g., variable name with an action.

<variable list> contains either a <characteristic> OF <object> or a <variable>.

Note: Only one alphabetic can be included in a single file statement.

Examples

Examples of using the FILE/ENDFILE verb are shown below. For clarification of output, it is recommended that only one or two variables be listed within a single file statement.

Example 1

In this example, the user wishes to display the time at which an error occurred within the simulation, the action name, and the error number.

```
IF number OF error NOT_EQUAL_TO 0 THEN
    FILE sim_time, action a contains error:
    number OF error ENDFILE
ENDIF
```

would result in the following statement in the output file:

```
00:00:05.100 action a contains error: 5
```

Example 2

The next example illustrates displaying the current value of an object characteristic.

```
FILE
    object a:
            char1= char1 of object_a
            char2= char2 of object_a
ENDFILE
```

This statement would result in the following statement in the output file:

```
object a: char1=on char2=23.46
```

The example below shows:

Example 3

```
IF status OF object_a NOT_EQUAL_TO okay
THEN
    FILE object a: ENDFILE
    FILE status= status of object_a ENDFILE
    FILE time_on= time_on of object_a ENDFILE
ENDIF
```

Output would be generated only if the status of object_a was not equal to okay. If the condition is true, the resulting output is:

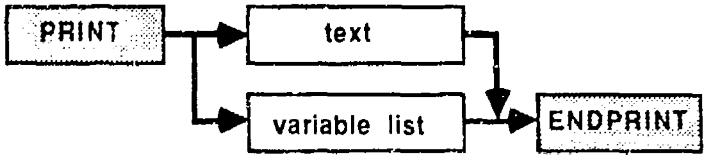
```
object a:
status=red_alert
time_on=43
```

The PRINT/ENDPRINT Verb

The PRINT/ENDPRINT verb is used to immediately print simulation values. Each PRINT statement results in a single line being sent to the printer. The statements will be printed sequentially based upon their occurrence within the simulation. The user can include explanatory text strings. The items included within the PRINT statement may be separated by commas or spaces and can occur in any order. Each PRINT statement must be paired with a terminating ENDPRINT. **NOTE:** if the print statement is used, the user's printer must be on and prepared to print.

Syntax

The syntax of the PRINT verb is shown below:



where:

<text> contains any series of alphanumeric characters that is not a name of a HOS item, e.g., variable name.

<variable list> contains either a <characteristic> OF <object> or a <variable>.

Examples

Examples of using the PRINT verb are shown below:

Example 1

The example below illustrates the user's request for an immediate printout when a critical value is reached in a radar system.

```
PRINT
processing an alert/warning symbol
action a, sim_time, symbol color= , color OF
symbol
ENDPRINT
```

The example above would send the following information to the printer:

```
processing an alert/warning symbol action a
    01:02:03.000 symbol color= violet
```

Example 2

The example below generates printer output when the condition in the IF statement is true:

```
IF status OF object_a NOT_EQUAL_TO okay
THEN
    PRINT
            object a:
                    status= status of object_a
                    char2= char2 of object_a
    ENDPRINT
ENDIF
```

The print line would be the following:

```
object a: status=off char2=green
```

RULE MANIPULATION VERBS

Rule manipulation verbs are used to add and remove rules from the active rule lists. Only active rules are evaluated during each simulation time interval. The START verb start rules while the SUSPEND verb suspends rules.

The START Verb

The START verb is used to add rules to the active rule lists. When a rule is initiated, the IF and UNTIL clauses are evaluated to determine if the rule will be activated. A specific rule can be started or an entire group of operator rules can be started. Operator rule groups are identified by the same first digit, i.e., rules 100 to 199 would be group 1. Additionally, all rules for a type can be started by specifying a rule type — environment, hardware, or operator.

Starting a Single Rule

The syntax of the START verb to start a single rule is shown below:

```
START <rule type> <rule number>
```

where:

<rule type> is ENVIRONMENT, HARDWARE or OPERATOR.

<rule number> is the rule number in the range of 0 to 99 for environment and hardware; 0 to 999 for operator.

Starting All Rules

The syntax of the START verb to start all rules for a type is shown below:

```
START <rule type>
```

where:

<rule type> is ENVIRONMENT, HARDWARE, or OPERATOR.

Example 1

The following illustrates starting all hardware rules:

```
START HARDWARE
```

Example 2

Entire groups of operator rules can be started by using the following syntax:

```
START OPERATOR GROUP <group number>
```

where:

<group number> is a single digit in the range of 0 to 9.

Example 3

Examples of using the START verb for operator groups are shown below:

```
START OPERATOR GROUP 9
START OPERATOR GROUP 6
```

The SUSPEND Verb

The SUSPEND verb is used to temporarily suspend the referenced rule from simulation processing. The rule number is preceded by the word ENVIRONMENT for environment rules; OPERATOR for operator rules; or HARDWARE for hardware rules. Use of a single rule type, e.g., ENVIRONMENT, HARDWARE, or OPERATOR, suspends all rules in the named category from the active rule lists.

Suspending a Single Rule

The syntax of the SUSPEND verb to suspend a single rule is shown below:

```
SUSPEND <rule type> <rule number>
```

where:

<rule type> is ENVIRONMENT, HARDWARE, or OPERATOR

<rule number> is the rule number in the range of 0 to 99 for environment and hardware; 0 to 999 for operator.

Note: Whenever the SUSPEND verb is used to deactivate an operator rule and that rule calls an action which accesses HOS micromodels, the following statement should also be executed:

```
DO HMM_micromodel_suspend
```

Example 1

Examples of suspending single rules are shown below:

```
SUSPEND OPERATOR 103
SUSPEND HARDWARE 87
SUSPEND ENVIRONMENT 63
```

Suspending All Rules

The syntax of the SUSPEND verb to suspend all rules for a type is shown below:

```
SUSPEND <rule type>
```

where:

<rule type> is ENVIRONMENT, HARDWARE, or OPERATOR.

Example 2

The following illustrates suspending all environment rules:

```
SUSPEND ENVIRONMENT
```

Suspending Groups of Operator Rules

Entire groups of operator rules can be suspended by using the following syntax:

```
SUSPEND OPERATOR GROUP <group number>
```

where:

<group number> is a single digit in the range of 0 to 9.

Example 3

Examples of using the SUSPEND verb to suspend all operator rules in group six (i.e., rules 600 through 699) is shown below:

```
SUSPEND OPERATOR GROUP 6
```

OBJECT MANIPULATION VERBS

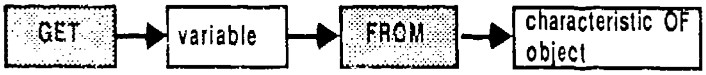
Object verbs are used to retrieve and manipulate the values of object characteristics. They are also used to locate and access members of object sets. The GET verb is used to access single object data while the RETRIEVE verb is used to access object sets. The PUT verb is used to modify the value of object characteristics. Brief examples of each verb are provided with each verb description. However, since the use of the GET, PUT, and RETRIEVE verbs are closely related, examples of their joint usage will be provided at the end of the section.

The GET Verb

The GET verb is used to retrieve the current value of a characteristic of an object and store that value in a variable for subsequent processing in the action. The GET verb can only be used on single objects or local objects that contain a single member of an object set. For object sets, a local object would be specified using the RETRIEVE verb prior to the GET. A PUT verb is required in order to store the contents of the variable back into the characteristic value.

Syntax

The syntax of the GET verb is shown below:



where:

<variable> is the name of a variable within the action that has been defined as a ALPHABETIC, DECIMAL number, or WHOLE number.

<characteristic> is the name of the object characteristic that has been defined within the action as a CHARACTERISTIC.

<object> is the name of a single object that has been defined within the action as an OBJECT or the name of a member of an object set that has been defined as a LOC_OBJECT.

Examples

An example of using the GET verb is shown below:

```
DEFINITIONS
    DECIMAL x_location
    WHOLE current_state
    OBJECT right_hand, object_a
    CHARACTERISTIC x_position, status
ENDDEFINITIONS
GET x_location FROM x_position OF right_hand
```

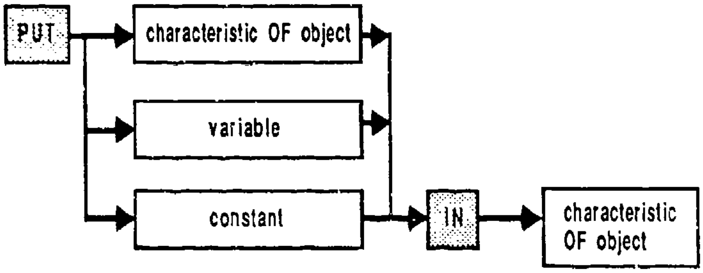
The GET statement illustrated above would store the current value of the x_position characteristic of the right_hand object in the local decimal variable x_location.

The PUT Verb

The PUT verb is used to store a value in the characteristic of an object.

Syntax

The syntax of the PUT verb is shown below:



where:

<send-value>is a <characteristic> OF <object>, <constant>, <variable>, or <alphabetic>.

<characteristic> is the name of the object characteristic that has been defined within the action as a CHARACTERISTIC.

<object> is the name of a single object that has been defined within the action as an OBJECT or the name of a member of an object set that has been defined as a LOC_OBJECT.

Examples

Examples of using the PUT verb are shown below:

```
DEFINITIONS
    CHARACTERISTIC status, x_location
    DECIMAL new_x_location
    OBJECT an_object, object_a, object_b
ENDDEFINITIONS
PUT on IN status OF an_object
PUT 0 IN x_location OF an_object
PUT new_x_location IN x_location OF an_object
PUT x_location OF object_a IN x_location OF
object b
```

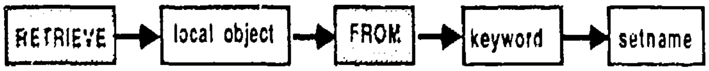
The first PUT would store the alphabetic on in the status characteristic of an_object. The second PUT would store 0 (zero) in the x_location characteristic of an_object. The third PUT would store the current value the decimal variable, new_x_location, in the x_location characteristic of an_object. The last PUT would store the current value of the x_location characteristic of object_a in the x_iocation characteristic of o'

The RETRIEVE Verb

The RETRIEVE verb is used to indicate which object in a defined set of objects is to be processed. The RETRIEVE verb places the desired object into a local object. The local object must be defined as a LOC_OBJECT in the DEFINITIONS section of the action. Once the appropriate member of the set is retrieved, then the GET and PUT verbs can be used to manipulate the characteristic values. The RETRIEVE verb contains keywords that permit the access of any specific member of the set, i.e., the first member, the last member, the current member, the previous member, and the next member. Additional relationships have also been defined for manipulating object sets—ENDOFSET can be used to test if the pointer is outside the boundaries of the objects set; NOTENDOFSET can be used to test if the pointer is within the boundaries of the object set.

Syntax

The syntax of the RETRIEVE verb is shown below:



where:

<local-object> is the variable name that will be used for a member of an object set and has been defined as a LOCAL_OBJECTS in the DEFINITIONS block.

<set-keyword> is one of the following: CURRENT, FIRST, LAST, NEXT, PREVIOUS, or a <whole number variable> (described below).

<set-name>is the name of an object set and defined as a SET in the DEFINITIONS block.

The keywords used to reference specific members of an object set are as follows:

CURRENT refers to the most recent referenced object in the set;

FIRST refers to the first object in the set;

LAST refers to the last object in the set. E.g., if the set contains 10 objects, **LAST** would reference object number 10;

NEXT references the next sequential member of the set;

PREVIOUS references the previously referenced member in the set; and

<local> is a local whole variable or constant that is used to directly reference a member of the set. E.g., if the sixth member of the set is to be referenced, 6 would be specified.

Set_name NOTENDOFSET is used to determine if the set reference is still within the boundaries of the set but within the conditionals IF and WHILE (see Conditional Verbs). Set_name ENDOFSET is used to determine if the set reference is outside the boundaries of the set but within conditionals IF and WHILE.

Object Manipulation Verb Examples

Examples of using the GET, PUT, and RETRIEVE verb are shown below:

```
DEFINITIONS
    WHOLE x_location, x_increment
    CHARACTERISTIC x_position
    OBJECT right_hand
ENDDEFINITIONS
GET x_location FROM x_position OF right_hand
SET x_location TO [x_location + x_increment]
PUT x_location IN x_position OF right_hand
```

This example shows how to obtain the current value of x_position of the right_hand object and put it in the x_location variable. After the SET statement, the PUT verb changes the value of the x_position characteristic of the right_hand object to the value of x_location.

Examples of using the RETRIEVE verb are shown below:

```
DEFINITIONS
    LOCAL_OBJECT an_emitter
    SETS emitters
    CHARACTERISTICS time
ENDDEFINITIONS
RETRIEVE an_emitter FROM FIRST emitters
WHILE emitters NOTENDOFSET THEN
    PUT sim_time IN time OF an_emitter
    RETRIEVE an_emitter FROM NEXT emitters
ENDWHILE
```

The example above illustrates the use of the RETRIEVE verb to loop through the members of the EMITTERS object set. As each member of the EMITTERS set is accessed, the contents are available in the local object named an_emitter. The WHILE verb (discussed in the Conditional Verb section) loops through all the emitters and puts the current simulation time in the time characteristic. Note the use of the first RETRIEVE that indicates the first member of the emitter set is to be retrieved. The RETRIEVE in the while is used to indicate that the next emitter is to be retrieved. Without the use of the second RETRIEVE, the first emitter would also be used.

COMPUTIONAL VERBS

The computational verb, SET, is used to modify the value of a variable to a constant or mathematical expression.

The SET Verb

The SET verb is used to change the value of a variable (i.e., the variable must have been previously defined as an ALPHABETIC, WHOLE or DECIMAL within the DEFINITIONS block). For whole and decimal numbers, it is also used to express formulas that perform calculations on variables. The formula must be enclosed in square brackets, i.e., [and]. Pairs of right and left parentheses can be used to indicate the order in which the formula is to be evaluated.

The following arithmetic operators are available:

- + indicates addition;
- - indicates subtraction;
- * indicates multiplication; and
- / indicates division.

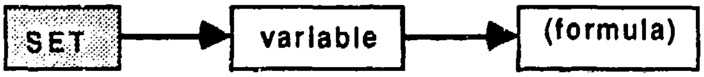
The standard mathematical functions that are available include:

- ACOS(x) — arc cosine of x in the range of 0 to π. The value of x must be between -1 and 1.
- ASIN(x) — arc sine of x in the range of -π/2 to π/2. The value of x must be between -1 and 1.
- ATAN(x) — arc tangent of x. ATAN returns a value in the range of -π/2 to π/2.
- ATAN2(x,y) — arc tangent of y/x. ATAN returns a value in the range of -π to π. The values of x and y cannot both be zero.
- COS(x) — cosine of x.
- FABS(x) — returns the absolute value of x.

- LOG(x) — calculates the natural logarithm of x. X must be a positive value greater than 0.
- LOG10(x) — calculates the base 10 logarithm of x. X must be a positive value greater than 0.
- POW(x,y) — computes x raised to the yth power. If x is zero and y is nonpositive or if x is negative and y is not an integer, an error will be returned.
- RAND() — returns a pseudo-random integer in the range of 0 to 215 - 1
- SEED(x) — sets the seed of the random number generator where x is between 0 and 9.
- SIN(x) — sine of x.
- SQRT(x) — calculates the square root of x. X must be a positive number.
- TAN(x) — tangent of x.

Syntax

The syntax of the SET verb is shown below:



<variable> is the name of a variable within the action that has been defined as an ALPHABETIC, DECIMAL, or WHOLE.

<formula> is a constant or mathematical statement enclosed within parentheses.

Example 1

The example below increments the current value of the time whole number variable by 1:

```
DEFINITIONS
   WHOLE time
ENDDEFINITIONS
SET time TO [time + 1]
```

Example 2

The example below sets the value of the decimal number variable, distance, to the square root of x1 times x2 plus y1 times y2 plus z1 times z2:

```
DEFINITIONS
   DECIMAL distance, x1, x2, y1, y2, z1, z2
ENDDEFINITIONS
SET distance TO [SQRT (x1*x2 + y1*y2 + z1*z2)]
```

Example 3

The example below stores the alphabetic off in the alphabetic variable current_status:

```
DEFINITIONS
   ALPHABETIC current_status
ENDDEFINITIONS
SET current_status TO [on]
```

Example 4

The example below sets the value of x to the following:

$$x = \frac{a+b}{2*c^3}$$

```
DEFINITIONS
   DECIMAL x,a,b,c
ENDDEFINITIONS
SET x to [(a+b)/(2*POW(c,3))]
```

CONDITIONAL VERBS

Conditional verbs are used for controlling the execution of a series of statements based upon the truth of a condition. Two conditional verbs are provided:

- IF condition THEN and
- WHILE condition THEN

Both the IF and WHILE verbs use expressions to specify the condition. Expressions consist of two items separated by a logical operator in the following form:

<item> logical operator <item>

where:

<item> is a constant, alphabetic, variable, or characteristic of an object and

logical operator is one of the following:

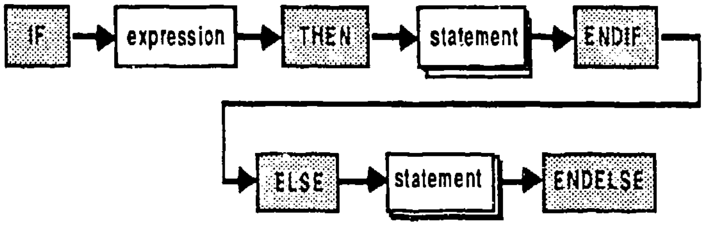
EQUALS
GREATER_OR_EQUAL
GREATER_THAN
LESS_OR_EQUAL
LESS_THAN
NOT_EQUAL_TO
ENDOFSET
NOTENDOFSET.

The IF Verb

The IF verb is used for controlling the execution of a series of statements based upon whether the expressed condition is true or not. If the condition is true, then the block of statements between THEN and ENDIF will be processed. Otherwise, program control will pass to the statement immediately following the ENDIF. If an alternative block of statements are to be processed the condition is false, then the optional ELSE <statement-group> ENDELSE statements are specified. IF verbs can be nested. However, the expression in each IF statement must be followed by the word THEN, a series of statements, the word ENDIF, and optionally ELSE ... ENDELSE. Each IF statement must be paired with a terminating ENDIF. Similarly, each ELSE statement must be paired with a terminating ENDELSE. Only a single condition can be expressed in an IF statement (i.e., AND and OR conditions are not permitted).

Syntax

The syntax of the IF verb is shown below:



where:

<expression> is a condition in the form <item> logical operator <item> as described in the definition of an expression and

<statement-group> is a series of statements that can contain any HOS verbs.

Example 1

The first example illustrates a simple if block that prints a message whenever a red symbol is encountered:

```
DEFINITIONS
    CHARACTERISTIC color
    OBJECT symbol
ENDDEFINITIONS
IF color OF symbol EQUALS red THEN
    PRINT processing alert at time, sim_time
    ENDPRINT
ENDIF
```

Example 2

This example illustrates the use of an alternative set of statements to be executed when the condition is using the ELSE verb:

```
DEFINITIONS
    CHARACTERISTIC color
    OBJECT symbol
ENDDEFINITIONS
IF color OF symbol EQUALS red THEN
    PRINT processing alert at time, sim_time
    ENDPRINT
ENDIF
ELSE
    IF color OF symbol EQUALS yellow THEN
            PRINT processing warning at time,
            sim_time
                    ENDPRINT
    ENDIF
    ELSE
            PRINT processing normal symbol,
            sim_time
            ENDPRINT
    ENDELSE
ENDELSE
```

If the color of the symbol is not red then the statements following the ELSE will be processed. These statements determine if the color of the symbol is yellow. If the symbol is not yellow then the statements following the next ELSE would be processed and the 'processing normal symbol' message will be printed.

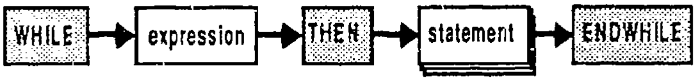
The WHILE Verb

The WHILE verb is used for controlling the execution of a series of statements as long as the condition is true. Each WHILE statement must be paired with a terminating ENDWHILE. If the condition is true, then the statements between THEN and ENDWHILE will be processed. Otherwise, program control passes to the statement immediately following the ENDWHILE. WHILE verbs can be nested; however, the expression in each WHILE statement must be followed by the word THEN, a series of statements, and the word ENDWHILE. The EXITWHILE verb can be used within the WHILE loop to exit to the statement immediately following the ENDWHILE. The expression for the WHILE verb is in the form <item> logical operator <item> as previously described. Only a single condition can be expressed in a WHILE statement (i.e., AND and OR conditions are not permitted).

The EXITWHILE verb is used to exit a WHILE loop and continue processing with the statement immediately following the ENDWHILE statement.

Syntax

The syntax of the WHILE verb is shown below:



where:

<expression> is a condition in the form <item> logical operator <item> as described in the definition of an expression and

<statement-group> is a series of statements that can contain any HOS verbs.

Example 1

The example below illustrates looping through a set of objects until the number of objects (noobject) is greater than the size of the object set (object_size). The last_time characteristic of an_object is modified to contain the current simulation time during each cycle through the WHILE loop.

```
DEFINITIONS
    WHOLE noobject, x_location, object_size,
            y_location
    DECIMAL last_time
    OBJECT an_object
    CHARACTERISTIC last_time
ENDDEFINITIONS
GET object_size FROM size OF my_objects
SET noobject TO [1]
WHILE noobject LESS_OR_EQUAL object_size
THEN
    PUT sim_time IN last_time OF an_object
    SET noobject TO [noobject + 1]
ENDWHILE
```

Example 2

This example illustrates using a WHILE statement to access various members of an object set.

```
DEFINITIONS
    WHOLE noobject, object_size
    DECIMAL x_location, y_location, last_time
    LOC_OBJECT an_object
    SETS object_a
    OBJECT screen_a
    CHARACTERISTIC status, x_position,
    y_position
ENDDEFINITIONS
RETRIEVE an_object FROM FIRST object_a
WHILE object_a NOTENDOFSET THEN
    IF status OF an_object NOT_EQUAL_TO on
            THEN
                    EXITWHILE

    ENDIF
    GET x_location FROM x_position OF
            an_object
    GET y_location FROM y_position OF
            an_object
```

```
        USING x_location,y_location
                DO calculate_distance
        PUT sim_time IN last_time OF an_object
        RETRIEVE an_object FROM NEXT object_a
    ENDWHILE
    COMMENT
        Continue processing here if the status of
        object_a is not on
    ENDCOMMENT
```

This example illustrates using the RETRIEVE verb to loop through all of the members of an object set. The status of each object is evaluated. If the status is not on, the EXITWHILE verb is used to exit the WHILE loop. In this case, the COMMENT statement immediately after the ENDWHILE would be executed next. If the status is on, then the object values are manipulated and a second RETRIEVE is used to access the next object in the set. The condition in the WHILE statement would again be evaluated to determine if the last member of the object_a set has been processed.

ACTION MANIPULATION VERBS

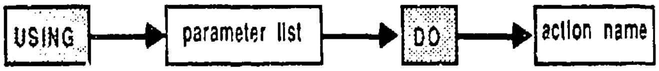
Action manipulation verbs are used to call other actions. The action manipulation verbs include USING...DO to call other actions with the optional ability to pass parameters and RECEIVE to obtain parameters from calling actions. Examples of using the USING...DO and RECEIVE verbs together are provided at the end of the section.

THE USING/DO VERB

The USING verb is used to invoke another action. Optional parameters can be used to pass values between the calling and the invoked action. The parameters can be passed in any order and are separated by spaces or commas. The user is responsible for ensuring that the invoked action correctly types the parameters with the appropriate DEFINITIONS. If no parameters are required, then only DO action name is necessary.

Syntax

The syntax of the USING/DO verb is shown below:



where:

<parameter> is a list of parameters to be used in the named action. The parameters can be object names, local variables or constants,

<action name> is the name of an action.

EXAMPLE 1

The example below:

DO action_1

would invoke action_1 without passing parameters.

EXAMPLE 2

This example would pass values contained in the size and color variables to the process_symbol action.

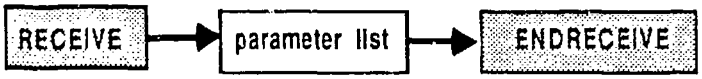
```
DEFINITIONS
    ALPHABETIC COLOR
    WHOLE SIZE
ENDDEFINITIONS
    GET color FROM symbol_color OF object_a
    SET size TO [2]
USING size, color DO process_symbol
```

The RECEIVE/ ENDRECEIVE Verb

The RECEIVE/ENDRECEIVE verb is used to indicate the name of the variables which are passed to an action via the parameters of the USING verb. The parameters must be received in the same order as specified in the USING statement. The parameters are separated by spaces or commas. The user is responsible for ensuring that the USING arguments and RECEIVE variables agree as to the type specified in the DEFINITIONS clause. Each RECEIVE statement must be paired with a terminating ENDRECEIVE.

Syntax

The syntax of the RECEIVE verb is shown below:



where:

<parameter> is a list of object names or local variables.

Processing Verb Example

An example of using the RECEIVE and USING verbw is shown below. The following shows the USING statement that passes the values of x_obj_location, y_obj_location, and z_obj_location to the action named distance_to_object:

```
DEFINITIONS
   DECIMAL x_obj_location, y_obj_location,
   z_obj_location
ENDDEFINITIONS
USING x_obj_location, y_obj_location,
   z_obj_location
   DO distance_to_object
```

The action named distance_to_object would contain the following:

```
COMMENT
   This routine calculates the distance from the
   center of the screen to an object location.
```

```
ENDCOMMENT
DEFINITIONS
   DECIMAL x_location,y_location,z_location
         x_screen,y_screen,z_screen
         x_dist,y_dist,z_dist
         distance
   OBJECT distance_calculator
   CHARACTERISTIC result
ENDDEFINITIONS
RECEIVE
   x_location,y_location,z_location
ENDRECEIVE
SET x_dist TO [x_location - x_screen]
SET y_dist TO [y_location - y_screen]
SET z_dist TO [z_location - z_screen)]
SET distance TO
   [SQRT (x_dist*x_dist + y_dist*y_dist +
         z_dist*z_dist))]
PUT distance IN result OF distance_calculator
END
```

This example also illustrates that the name of the parameters used to invoke an action do not have to match those in the receiving action.

SPECIAL VERBS

HOS-IV contains provision for including models written using the C language (Microsoft C Version 4.0). These models can be directly integrated into the HOS actions. The START_C_CODE and END_C_CODE verbs are used to enclose the C code.

The START_C_CODE END_C_CODE Verb

The START_C_CODE verb is used to embed C code within an action. The C code must be written to conform to the standards of Microsoft C compiler Version 4.0. The END_C_CODE verb must immediately follow the end of the C code. All variables included in the C code most be identified by type in the DEFINITIONS verb.

Syntax

The syntax of the START_C_CODE verb is:

```
START_C_CODE
    user's C code
    END_C_CODE
```

Examples

Examples of using the START_C_CODE verb are shown below:

```
START_C_CODE
    User's C code
END_C_CODE
```

TERMINATING VERBS

The terminating verbs are used to terminate an action (END) and terminate the simulation (END_SIM).

The END Verb

The END verb is used to indicate the end of an action and must be placed as the last statement in each action. The END statement must be terminated with a carriage return.

Syntax

The syntax of the END verb are:

```
END
```

Examples

Examples of using the END verb is:

```
COMMENT This is an action ENDCOMMENT
statement
statement
statement
END
```

The END_SIM Verb

The END_SIM verb is used to stop the simulation. It can be included in any action and may be used more than once to trigger various terminating conditions. If the END_SIM verb is not used to terminate the simulation, then the simulation will continue until the end simulation time defined in Setup Simulation (see Section 5) is reached.

Syntax

The syntax of the END_SIM verb is:

```
END_SIM
```

Examples

Examples of using the END_SIM verb are shown below:

```
IF sim_time GREATER_THAN 10000 THEN
    PRINT
            simulation time exceeded, sim_time
    ENDPRINT
    END_SIM
ENDIF
ELSE
    IF number OF error EQUALS 5 THEN
            PRINT error condition = 5. at time = ,
            sim_time ENDPRINT
            END_SIM
    ENDIF
ENDELSE
```

In this example, the END_SIM verb is used to terminate the simulation if the sim_time is greater than 10000 time units or if error number 5 is encountered.

8. HOS MICROMODELS

This section presents a general description of each HOS micromodel including underlying concepts and algorithms. References are provided for each micromodel as well as examples of use. Following the descriptions of the micromodels, methods for modifying existing micromodels or building new micromodels is presented. A discussion of the two modes of micromodel operation — sequential and parallel — is presented at the conclusion of this section.

MicroModel Access

The micromodels are automatically incorporated into a simulation if the Use Micromodels option is selected when defining the HOS simulation parameters (see Section 5). Specifically, at simulation start, when this option is selected, several actions will be executed to start all the rules associated with the micromodels as well as convert any micromodel timing parameters from the default of seconds to the user selected simulation time unit.

All micromodels are written using the HOS Action Language (HAL) (as described in Section 7) and are stored as separate HOS actions. These models can be accessed during the simulation by either a DO or a USING...DO verb. The DO verb is used when the micromodel does not require any input parameters; the USING...DO verb is used when the micromodel requires specification of parameters from the calling action. For example:

DO *micromodel_action_name*

USING *input 1, input 2,...input n*
    DO *micromodel_action_name*

MicroModel Actions

The names of all micromodel actions begin with the prefix HMM (HOS MicroModel). The HMM actions can be viewed within the **Action Editor** by selecting the **Open** option from the **File** menu and using the mouse to click on the name of the desired micromodel.

MicroModel Rules and Objects

The rules and objects associated with HOS micromodels also begin with the prefix HMM and are viewable from the **Rule** and **Object Editors**, respectively. Micromodel objects store the values of key parameters that affect the timing and accuracy of the particular human behavior being simulated. The timing parameters stored in these objects are defined in time units of seconds (with the exception of the fatigue model). Micromodels rules are typically clocks which affect the length of time over which a particular behavior will occur during the simulaiton. Micromodel rules are assigned to Operator Rule Group 0 and are automatically incorporated into each HOS simulation.

Simulation Time Units and MicroModels

Most micromodel timing parameters are stored in the default time unit of seconds in the object data base. The HOS micromodels can, however, be used with any simulation independent of the simulation's defined time unit. When the USE MICROMODELS option is selected in the SIMULATION SETUP editor, HOS will automatically convert the micromodel timing parameters (stored in the object data base as seconds) to the user selected time unit. Thus, if eye movement time is stored as 0.17 seconds, at simulation start, HOS will automatically convert this number to the user selected simulation time unit. If the simulation time unit is tenths of a second, this number is converted to 1.7 tenths; if the simulation time unit is hundredths of a second, this number is converted to 17 hundredths. For the latter case, this means that an eye movement will occur over 17 simulation time intervals.

Note: If the default micromodel parameters are modified directly in the object data base, the new value must be entered using the default of seconds.

Using MicroModels

When building a simulation, the user must define the sequence of behavioral steps that will occur in a particular action. Most actions do not consist of a single step (e.g., an eye movement alone) but rather consist of a number of sequential steps. For example, an action for a display monitoring task will consist of a series of behavioral steps:

- Move eyes to the display;
- Visually perceive the necessary data;
- Make a decision about the status of the system;
- Perhaps move the hand to a control, and;
- Finally adjust or manipulate a particular control.

For each action, the micromodel calls required for the particular task must be listed. A sample action — named SAM_main_monitor — of an operator performing a monitoring task is shown in Figure 8-1. Associated with this action is an object (SAM_main_monitor_status) that tracks the step currently being performed during the simulation as well as the micromodel in use. For each action which contains a sequential series of behavioral steps, the user must use the Object Editor to define a step-tracking object associated with the action. This object must be defined with two characteristics (Note: additional characteristics may also be defined):

1. STEP -- tracks which step number is being performed (usually initialized to 1), and
2. MICROMODEL_IN_USE -- tracks which micromodel is being performed at each simulation time.

The user must pass the name of this object to each micromodel (with the exception of the fatigue model, see below), along with any other required input parameters. When a particular behavior is complete, the HOS micromodel will automatically update the STEP characteristic of this object so that the next sequential behavior can then be performed.

```
COMMENT                    SAM_main_monitor
  This action is called by the OPERATOR RULE 800 to simulate the
  operator performing substeps:
          1.      fixate on reference
          2.      visual perception of information contained in reference
          3.      hand movement
ENDCOMMENT
DEFINITIONS
OBJECT       SAM_main_monitor_status
CHARACTERISTIC  step
DECIMAL      distance, width
ENDDEFINITIONS
IF step OF SAM_main_monitor_status EQUALS 1 THEN
      USING SAM_main_monitor_status DO HMM_eye_movement
ENDIF
IF step OF SAM_main_monitor_status EQUALS 2 THEN
      USING SAM_main_monitor_status DO HMM_visual_perceive
ENDIF
IF step OF SAM_main_monitor_status EQUALS 3 THEN
      SET distance TO [10.]
      SET width TO [0.5]
      USING SAM_main_monitor_status, distance, width
            DO HMM_right_hand_movement
ENDIF
IF step OF SAM_main_monitor_status EQUALS 4 THEN
      PUT 1 IN step OF SAM_main_monitor_status
ENDIF
END
```

Figure 8-1. Sample HOS Action to Simulate Sequential Human Behavior

Within this section, this action-related object which tracks the current micromodel being executed as well as the current step is referred to as *calling_action_object*. The specific name of this object will vary depending on the name of the object that the user has uniquely defined for each action. For consistency, it is recommended that the name of the object be constructed by combining the name of the action with the suffix _STATUS.

DECISION TIME

The decision time micromodel determines the time required for the simulated operator to select one of many alternatives during a simulation.

Description

The decision time model is based on Hick's Law (1952). The model assumes that decision time increases as the number of alternatives (or uncertainty) increases. Thus,

$$\text{Decision Time} = k * \log_2 (n + 1)$$

where:

k is a constant representing simple reaction time and

n is the number of possible alternatives.

A default value of 150 milliseconds was selected for the constant k as recommended by Card, et al. (1986).

HAL Access Statement

The decision time micromodel is invoked in the following way:

```
USING calling_action_object,
    number_alternatives
DO HMM_DECISION
```

where:

calling_action_object is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section).

number_alternatives is a whole number variable which defines the number of equally probable alternatives.

Decision Micromodel Example

A sample of a user-defined action which calls HMM_DECISION is given below:

```
DEFINITIONS
    OBJECT calling_action_object
    CHARACTERISTIC step
    WHOLE number_alternatives
ENDDEFINITIONS
    [user code]
SET number_alternatives TO [9]
USING calling_action_object,
    number_alternatives
DO HMM_decision
```

Execution of the action HMM_DECISION will result in a simulation time charge which will vary depending on the number of choice alternatives input by the user. The default time charge per choice (the constant k) is stored in the DECISON_CONSTANT characteristic of the HMM_DECISION_STATUS object.

References

Card, S.K., Moran, T.P., and Newell, A. (1986). The model human processor: An engineering model of human performance. In *Handbook of Perception and Human Performance, Volume II*, K.R. Boff, L. Kaufman, and J.P. Thomas, (Eds.) New York, NY: Wiley-Interscience, p. 45-7.

Hick, W.E. (1952). On the rate of gain of information. *Quarterly Journal of Experimental Psychology*, 4, 11-26.

EYE MOVEMENT

The eye movement model determines the time required to move the eye to a new fixation point.

Description

The default time charge of 0.17 seconds for an eye movement is based on research conducted by Russo (1978). This time charge is the summation of a number of microsteps in an eye movement cycle. These steps include:

1. Determine location of fixation (0.05 seconds),
2. Transmit movement command to motor system (0.03 seconds),
3. Move eye (0.03 seconds), and
4. Transmit stimulus to cortex (0.06 seconds).

The total of these microsteps is 0.17 seconds.

HAL Access Statement

The eye movement micromodel is invoked in the following way:

```
USING calling_action_object
DO HMM_EYE_MOVEMENT
```

where:

calling_action_object is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section).

Execution of the action HMM_EYE_MOVEMENT results in a simulation time charge for the ballistic action of the eye moving from one fixation to the next. The time charge is stored in the TIME characteristic of the HMM_EYE_MOVEMENT_STATUS object.

Reference

Russo, J.E. (1978). Adaptation of cognitive processes to the eye movement system. In *Eye Movement and the Higher Psychological Functions,* J.W. Senders, D.F. Fisher, and R.A. Monty, (Eds.), Hillsdale, NJ: Lawrence Erlbaum Associates Publishers, pp. 89-112.

FATIGUE

The fatigue model updates the values of all HOS micromodel timing and accuracy parameters to simulate the effects of fatigue over time.

Description

A model to simulate fatigue effects was derived from an analysis of numerous studies that investigated decrements in both cognitive and motor functions over time. The model predicts the percentage decrement in performance for a given time t:

Increased performance time or error =
$1.0/[.25(.93)^t + .75]$

where:

t is the number of hours of steady performance.

This function was based upon results from several sources — particularly Oshima (1981). The function models the expected reduction in productive performance for periods of up to 24 hours (independent of diurnal variation). Thus, the model assumes 0 percent decrement at time 0; 2 percent decrment at one hour; 7 percent decrement at four hours; and a 12 percent decrement after eight hours.

HAL Access Statement

DO HMM_FATIGUE

HMM_FATIGUE will adjust micromodel parameters at five minute intervals to reflect changes in performance time and accuracies. The user only needs to call this model once. The five minute update will occur automatically. To affect the value of initial operator fatigue at simulation start, modify the characteristic CUMULATIVE_HOURS_FATIGUE of HMM_FATIGUE_STATUS (default is 0). To change the update interval, the UPDATE_INTERVAL characteristic of the HMM_FATIGUE_STATUS object can be modified directly using the OBJECT EDITOR. The default value is 300 seconds (five minutes). If the UPDATE_INTERVAL is changed, the

HOURS_PER_UPDATE_INTERVAL characteristic must also be changed. The default of the latter characteristic is 0.083 (that is, 0.083 hours in five minutes).

Reference

Oshima, M. (1981). The mechanism of mental fatigue. In *Machine Pacing and Occupational Stress*, G. Salvendy & M. Smith (Eds.) London: Taylor & Francis, pp. 81-89

HAND MOVEMENT

HOS provides two hand movement models which work independently — one for the right hand and a second for the left hand. The right hand is modeled using the HMM_RIGHT_HAND_MOVEMENT action; while the left hand uses HMM_LEFT_HAND_MOVEMENT action. Since both models utilize identical algorithms, the discussion below applies to both models.

Description

The hand movement action determines the time required to move the hand to a new position. The movement time model is based on Fitts' Law (Fitts and Petterson, 1964) which states that movement time is a linear function of the information content or difficulty of movement:

$$\text{Hand movement time} = k * \log_2 (d / w + .5)$$

where:

k is a constant with a default setting of 0.1 seconds as recommended by Card, et al. (1986),

d is the distance to be moved, and

w is the width of the target which the hand is moving to.

HAL Access Statement

The hand movement micromodels are invoked as illustrated below:

```
USING calling_action_object, inches_to_move,
    width_in_inches
    DO HMM_whichhand_HAND_MOVEMENT
```

where:

calling_action_object is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section)

inches_to_move is a decimal variable which defines the distance in inches which the hand must travel (e.g., 10.2).

width_in_inches is a decimal variable which defines the width of the target in inches (e.g., 1.5).

whichhand indicates which hand — right or left.

Hand Movement Model Example

A sample of a user-defined action which calls HMM_RIGHT_HAND_MOVEMENT is given below:

```
DEFINITIONS
    OBJECT calling_action_object
    CHARACTERISTIC step
    DECIMAL inches_to_move, width_in_inches
ENDDEFINITIONS
    [user code]
SET inches_to_move TO [10.2]
SET width_in_inches TO [1.5]
USING calling_action_object, inches_to_move,
    width_in_inches
    DO HMM_RIGHT_HAND_MOVEMENT
```

Execution of HMM_RIGHT_HAND_MOVEMENT results in a simulation time charge required to move the right hand to a new position.

A sample of a user-defined action which calls HMM_LEFT_HAND_MOVEMENT is given below:

```
DEFINITIONS
    OBJECT calling_action_object
    CHARACTERISTIC step
DECIMAL inches_to_move, width_in_inches
ENDDEFINITIONS
    [user code]
SET inches_to_move TO [5.6]
SET width_in_inches TO [0.5]
USING calling_action_object, inches_to_move,
    width_in_inches
    DO HMM_LEFT_HAND_MOVEMENT
```

Execution of HMM_LEFT_HAND_MOVEMENT will result in a simulation time charge for moving the left hand to a new position.

References

Card, S.K., Moran, T.P., and Newell, A. (1986). The model human processor: An engineering model of human performance. In *Handbook of Perception and Human Performance, Volume II*, K.R. Boff, L. Kaufman, and J.P. Thomas, Eds. New York, NY: Wiley-Interscience, 1986.

Fitts, P.M. & Peterson, J.R. (1964). Information capacity of discrete motor responses. *Journal of Experimental Psychology*, 67, 103-112.

HANDPRINTING

The handprinting micromodel determines a time charge for handprinting (as opposed to handwriting) a specific number of characters.

Description

The default time charge for handprinting is 750 milliseconds per character (Devoe, 1967).

HAL Access Statement

The handprinting micromodel is invoked as shown below:

```
USING calling_action_object, number_of_char
DO HMM_HANDPRINT
```

where:

calling_action_object is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section).

number_of_char is a whole number variable which defines the number of characters tc be printed.

Handprinting Micromodel Example

A sample of a user-defined Action which calls HMM_HANDPRINT is given below:

```
DEFINITIONS
    OBJECT calling_action_object
    CHARACTERISTIC step
    WHOLE number_of_char
ENDDEFINITIONS
    [user code]
SET number_of_char TO [20]
USING calling_action_object, number_of_char
    DO HMM_HANDPRINT
```

Execution of the action HMM_HANDPRINT results in a simulation time charge which varies depending on the number of characters printed. The default time charge per character is stored in the time_per_character characteristic of the HMM_HANDPRINT_STATUS object.

Reference

Devoe, D.B. (1967). Alternative to handprinting in the manual entry of data. *IEEE Transactions on Human Factors in Electronics*, HFE-8, 21-31.

LISTENING

The listening micromodel determines the probability of a listening error based on the signal-to-noise ratio and the interruption frequency of the noise. A time charge for listening is also determined.

Description

The HMM_LISTEN action determines the probability of an error in listening for four levels of signal-to-noise ratios and three levels of noise interrupt frequency. The model is based on research conducted by Kryter (1970). The four levels of signal-to-noise which are simulated are: +9, 0, -9, and -18 decibels. The three levels of noise interruption frequency simulated are 1, 10, and 100 Hertz (cycles per second). A listening time charge of 2.4 words per second is assumed (Pierce and Karlin, 1957).

HAL Access Statement

The listening micromodel is invoked as shown below:

```
USING calling_action_object, number_of_words,
   level_SN, level_noise_freq
   DO HMM_LISTEN
```

where:

calling_action_object is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section).

number_of_words is a whole number variable that defines the number of words to be spoken to the simulated listener.

level_SN is a whole number variable which defines one of four signal-to-noise decibels levels: 9, 0, -9, or -18.

level_noise_freq is a whole number variable with one of three noise interrupt frequencies in HZ: 1, 10, or 100.

Listening Micromodel Example

A sample of a user-defined Action which calls HMM_LISTEN is given below:

```
DEFINITIONS
```

```
    OBJECT calling_action_object
    CHARACTERISTIC step
    WHOLE number_of_words, level_SN,
    level_noise_freq
ENDDEFINITIONS
    [user code]
SET number_of_words TO [25]
SET level_sn TO [-9]
SET level_noise_freq TO [10]
USING calling_action_object, number_of_words,
    level_SN, level_noise_freq
    DO HMM_LISTEN
```

Execution of the action HMM_LISTEN results in a time charge for listening based on the number of words the simulated operator is listening to. The micromodel also updates the RESULT characteristic of the HMM_LISTEN_ERROR_OUTPUT object. RESULT is an alphabetic variable which contains ERROR or NO_ERROR. The HOS user can then write a conditional statement which alters the course of the simulation based on the value of result. For example,

```
IF result of HMM_LISTEN_ERROR_OUTPUT
    EQUALS error THEN
    DO repeat_message
ENDIF
```

The default error rates for each signal-to-noise/noise interrupt combination are stored in the FRROR_RATE characteristic of the HMM_LISTEN_ERROR_DATA object set. The HOS model selects the appropriate probability of error based on the user-specified inputs. The following table lists the default setting.

	Set		Default
OBJECT SET	No.	CHARACTERISTIC	
	Value		
HMM_LISTEN_ERROR_DATA			
	1	SN_decibels	9
		noise_HZ	1
		error_rate	.10
	2	SN_decibels	9
		noise_HZ	10
		error_rate	.07

3	SN_decibels	9
	noise_HZ	100
	error_rate	.10
4	SN_decibels	0
	noise_HZ	1
	error_rate	.20
5	SN_decibels	0
	noise_HZ	10
	error_rate	.12
6	SN_decibels	0
	noise_HZ	100
	error_rate	.25
7	SN_decibels	-9
	noise_HZ	1
	error_rate	.35
8	SN_decibels	-9
	noise_HZ	10
	error_rate	.25
9	SN_decibels	-9
	noise_HZ	100
	error_rate	.65
10	SN_decibels	-18
	noise_HZ	1
	error_rate	.42
11	SN_decibels	-18
	noise_HZ	10
	error_rate	.28
12	SN_decibels	-18
	noise_HZ	100
	error_rate	.96

References

Kryter, K.D. (1970). *The Effect of Noise on Man.* New York: Academic Press, p.58.

Pierce, J.R., and Karlin, J.E. (1957). Reading rates and the information rate of a human channel. *Bell Telephone Technical Journal*, Vol. 36, pp. 497-516.

MANIPULATE CONTROL

HOS provides two control manipulation models which work independently — one for the right hand and a second for the left hand. The right hand is modeled using the HMM_RIGHT_HAND_MANIP action while the left hand uses the HMM_LEFT_HAND_MANIP action. Since both models utilize identical algorithms, the discussion below applies to both models.

Description

The HOS control model determines the simulation time charge for manipulating four different types of controls — pushbutton, toggle, rotary dial, or trackball. The user passes an input parameter to the model which identifies the specific control type to be manipulated. If the trackball type is selected, two additional inputs are required:

1. The distance the screen cursor must be moved and
2. The width of the target symbol.

For manipulating a pushbutton or toggle, 0.4 seconds will be charged to the simulation; (Goldbeck & Charlet, 1974) for manipulating a rotary dial, 0.73 seconds will be charged. The trackball model is based on Fitts' Law (Fitts & Petterson, 1964):

$$\text{Trackball movement time} = k * \log_2 (d/w + .5)$$

where:

k is a constant with a default setting of 0.1 seconds as recommended by Card, et al. (1986),

d is the distance the cursor is to be moved, and

w is the width of the symbol on the display.

HAL Access Statement

The manipulate control micromodel is invoked as shown below:

```
USING calling_action_object, control_type,
    inches_to_move, width_in_inches
    DO HMM_whichhand_HAND_MANIP
```

where:

calling_action_object is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section).

control_type is a alphabetic variable which defines the type of the control to b e manipulated. This parameter has four possible values — PUSHBUTTON, TOGGLE, ROTARY_DIAL, or TRACKBALL. Note: If the value of control_name is set to TRACKBALL, the following parameters must also be passed.

- **inches_to_move** (TRACKBALL case only) is a decimal variable which defines the distance in inches that the cursor must travel (e.g., 4.5) and
- **width_in_inches** is a decimal variable which defines the width of the symbol on the display in inches (e.g., 0.2).

whichhand indicates which hand – right or left.

Manipulate Control Micromodel Example

A sample of a user-defined Action which calls HMM_RIGHT_HAND_MANIP is given below:

```
DEFINITIONS
   OBJECT calling_action_object
   CHARACTERISTIC step
   ALPHABETIC control_type
   DECIMAL inches_to_move, width_in_inches
ENDDEFINITIONS
   [user code]
SET control_type TO [trackball]
SET inches_to_move TO [4.5]
SET width_in_inches TO [0.2]
USING calling_action_object, control_type,
   inches_to_move,
   width_in_inches
   DO HMM_RIGHT_HAND_MANIP
   [user code]
SET control_type TO [pushbutton)
USING calling_action_object, control_type DO
   HMM_RIGHT_HAND_MANIP
```

Execution of HMM_RIGHT_HAND_MANIP will result in a simulation time charge for the desired control action defined by the user.

A sample of a user-defined action which calls HMM_LEFT_HAND_MANIP is given below:

```
DEFINITIONS
    OBJECT calling_action_object
    CHARACTERISTIC step
    ALPHABETIC control_type
    DECIMAL inches_to_move, width_in_inches
ENDDEFINITIONS
    [user code]
SET control_type TO [trackball]
SET inches_to_move TO [4.5]
SET width_in_inches TO [0.2]
USING calling_action_object, control_type,
    inches_to_move,
    width_in_inches
    DO HMM_LEFT_HAND_MANIP
    [user code]
SET control_type TO [pushbutton]
USING calling_action_object, control_type DO
    HMM_LEFT_HAND_MANIP
```

References

Card, S.K., Moran, T.P., and Newell, A. (1986). The model human processor: An engineering model of human performance. In *Handbook of Perception and Human Performance, Volume II*, K.R. Boff, L. Kaufman, and J.P. Thomas, (Eds.) New York, NY: Wiley-Interscience, 1986.

Fitts, P.M. & Peterson, J.R. (1964). Information capacity of discrete motor responses. *Journal of Experimental Psychology*, 67, 103-112.

Goldbeck, R.A., and Charlet, J.D. (1974). *Task Parameters for Predicting Panel Layout Design and Operator Performance.* Philco_Ford Technical Report WDL-TR5480, Palo Alto, Philco-Ford.

SHORT TERM MEMORY

The short term memory model simulates a push down memory stack which stores a fixed number of items. New items stored in memory are placed at the top of the list; old items at the bottom of the list are eliminated as new items are entered. This model can be used to simulate the operator reading the values of display objects and placing the values in memory for use in the near future (i.e., within a minute after entry). The maximum size of the short term or working memory model (stored in the characteristic MAXIMUM_SIZE of the HMM_MEMORY_DATA object) has a default value of seven. The maximum memory size as well as the simulation time charge for the storage process (stored in the characteristic STORAGE_TIME of the HMM_MEMORY_DATA object) were adopted from default values recommended by Card, Moran, and Newell (1986). The default storage time is 0.1 seconds and is equivalent to the cognitive cycle time described by Card, et al. (1986).

Description

The short term memory or working memory model consists of two separate actions:

1. HMM_MEMORY_STORE and
2. HMM_MEMORY_RETRIEVE.

The store action is used to enter new items in memory. For instance, the user may want to simulate an operator remembering the values read from displays. For such cases, the user passes the name of the object and the name of the characteristic to be remembered as well as the value to be remembered to the HMM_MEMORY_STORE action. Similarly, when retrieving a data item, the name of both the object and characteristic are passed to the HMM_MEMORY_RETRIEVE action.

Working Memory Store

The HMM_MEMORY_STORE action stores information into short term memory.

HAL Access Statement

The HAL access statement varies depending on the particular case. Specifically, the number and type of parameters varies depending on what value must be stored. There are six cases:

1. Store an alphabetic value;
2. Store a decimal value;
3. Store a whole value;
4. Store an alphabetic value of an element of an object set;
5. Store a decimal value of an element of a set; and
6. Store a whole value of an element of a set.

The syntax for activation of HMM_MEMORY_STORE is as follows. Optional parameters are italicized. Mandatory parameters which vary by case are bracketed.

```
USING calling_action_object, name_object,
    name_char, type_of_item,
    [alphabetic_value OR decimal_value OR
    whole_value ],
    element_number
    DO HMM_MEMORY_STORE
```

where:

calling_action_object is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section).

name_object is an alphabetic variable which defines the name of the object to be stored in memory (e.g., ON_TRUE_BRG_DISPLAY). Note: The name of the object is constructed by entering the prefix ON_ to the real name of the object. This name must also be entered in the alphabetic dictionary of the object editor.

name_char is an alphabetic variable which defines the name of the characteristic of the object to be stored in memory (e.g., ON_bearing). Note: The name of the object is constructed by entering the prefix ON_ to the real name of the characteristic. This name must also be entered in the alphabetic dictionary of the object editor.

type_of_item is an alphabetic variable which defines the type of item to be stored in memory (e.g., type_d). Note: There are six possible values for this parameter:

VALUE	CASE
type_a	alphabetic
type_d	decimal
type_w	whole
type_a_set	alphabetic/set
type_d_set	decimal/set
type_w_set	whole/set

The last three options apply to items which are elements of a set. When one of the last three options is selected, the user must also pass the *element_number* parameter.

alphabetic_value is an alphabetic variable which holds the value to be stored. Note: This parameter is used when the value of type_of_item is TYPE_A or TYPE_A_SET.

decimal_value is a decimal variable which stores the value to be put in memory. Note: This parameter is used when the value of type_of_item is TYPE_D or TYPE_D_SET.

whole_value is a whole number variable which stores the whole value to be put in memory. Note: This parameter is used when the value of type_of_item is TYPE_W or TYPE_W_SET.

element_number is a whole number variable which stores the pointer to the specific item of a set which is to be remembered. Note: This parameter is used when the value of type_of_item is TYPE_A_SET, TYPE_D_SET, or TYPE_W_SET.

The syntax used to store a decimal value is as follows:

```
USING calling_action_object,
   name_object, name_char,
type_of_item, decimal_value
DO HMM_MEMORY_STORE
```

An example of storing a whole value of an element in an object set is as follows:

```
USING calling_action_object,
   name_object, name_char,
   type_of_item, whole_value,
   element_number
   DO HMM_MEMORY_STORE
```

Working Memory Store Example

A sample of a user-defined Action which calls HMM_MEMORY_STORE is given below:

```
DEFINITIONS
   OBJECT true_brg_display,
         calling_action_object
   CHARACTERISTIC bearing, step
   ALPHABETIC name_object, name_char,
         type_of_item
   DECIMAL decimal_value
ENDDEFINITIONS
   [user code]
SET name_object TO [ON_true_brg_display]
SET name_char TO [CN_bearing]
SET type_of_item TO [type_d]
GET decimal_value FROM bearing OF
   true_brg_display
USING calling_action_object, name_object,
   name_char, type_of_item,
   decimal_value
   DO HMM_MEMORY_STORE
```

The HMM_MEMORY_STORE action will push down the memory stack (the object set HMM_MEMORY) and enter the new item at the top of stack. A storage time will be charged to the simulation. The value for the storage time is stored in the characteristic named STORAGE_TIME of the object named HMM_MEMORY_DATA. The store action will also note the time of entry of items into the stack. The entry time will be used later by the HMM_MEMORY_RETRIEVE action to determine if the item can be remembered.

Reference

Card, S.K., Moran, T.P., and Newell, A. (1986). The model human processor: An engineering model of human performance. In *Handbook of Perception and Human Performance, Volume II*, K.R. Boff, L. Kaufman, and J.P. Thomas, Eds. New York, NY: Wiley-Interscience, 1986.

Working Memory Retrieve

HMM_MEMORY_RETRIEVE is used to simulate an operator searching through short term memory for an item which has been entered on the stack via the HMM_MEMORY_STORE action. The retrieve action searches the memory stack, charging the simulation for each item searched. The results of the search are entered in the HMM_ACTIVE_MEMORY_CELL object. If the particular item the operator is searching for is found, the time of entry of that item into memory is used to determine if the value can be remembered.

The first step in the memory retrieval process is simulation of the operator labeling the item to be retrieved. This step will result in a simulation time charge of 0.07 seconds (stored in the characteristic RETR_LABEL_TIME of the object named HMM_MEMORY_DATA). This value is based on research conducted by Posner (1978) which indicates that name matches take 70 milliseconds longer than physical matches. The actual retrieval search time has a default setting of 0.047 seconds per item and is based on data collected by Cavanaugh (1972) as reported by Card, Moran, and Newell (1986).

If the item being searched for is found, the HMM_MEMORY_RETRIEVE action will then determine if the item can be successfully recalled. The model default is a half-life of seven seconds for each item is memory. This means that seven seconds after entry into HMM_MEMORY, there is a 50 percent probability that the item will be recalled. Twenty-eight seconds after entry, the probability of recall is 25 percent, and so on. The default half life of seven seconds is based on a recommendation by Card, et al. (1986).

HAL Access Statement

The HAL access statement varies depending on the particular case. Specifically, the number and type of parameters will vary depending on what item is to be retrieved. There are six cases:

1. Retrieve an alphabetic value;
2. Retrieve a decimal value;

3. Retrieve a whole value;
4. Retrieve an alphabetic value of an element of an object set;
5. Retrieve a decimal value of an element of a set; and
6. Retrieve a whole value of an element of a set.

The syntax for activation of HMM_WORKING_MEMORY_RETRIEVE is as follows with optional parameters shown in italics:

```
USING calling_action_object, name_object,
    name_char, type_of_item, element_number
    DO HMM_MEMORY_RETRIEVE
```

where:

calling_action_object is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section).

name_object is an alphabetic variable which defines the name of the object to be retr'eved from memory (e.g., ON_true_brg_display).

name_char is an alphabetic variable which defines the name of the characteristic of the object to be retrieved from memory (e.g., ON_bearing)

type_of_item is an alphabetic variable which defines the type of the item to be retrieved from memory (e.g., type_d) Note: There are six possible values for type_of_item:

VALUE	CASE
type_a	alphabetic
type_d	decimal
type_w	whole
type_a_set	alphabetic/set
type_d_set	decimal/set
type_w_set	whole/set

The last three options apply to items which are elements of a set. When one of the last three options are selected, the user must also pass the **element_number** parameter.

element_number is a whole number variable which defines the pointer to the specific item of a set which is to be retrieved. Note: This parameter is used when the value of type_of_item is TYPE_A_SET, TYPE_D_SET, or TYPE_W_SET.

The syntax required to retrieve an alphabetic value is as follows:

```
USING calling_action_object,
   name_object, name_char, type_of_item
   DO HMM_MEMORY_RETRIEVE
```

An example of retrieving a decimal value of an element in an object set is shown below:

```
USING calling_action_object,
   name_object, name_char,
   type_of_item, element_number
   DO HMM_MEMORY_RETRIEVE
```

Working Memory Retrieve Example

A sample of a user-defined Action which calls HMM_MEMORY_RETRIEVE is given below:

```
DEFINITIONS
   OBJECT true_brg_display,
         calling_action_object
   CHARACTERISTIC bearing, step
   ALPHABETIC name_object, name_char,
         type_of_item
ENDDEFINITIONS
   [user code]
SET name_object TO [ON_true_brg_display]
SET name_char TO [CN_bearing]
SET type_of_item TO [type_d]
USING name_object, name_char, type_of_item
   DO HMM_MEMORY_RETRIEVE
```

Execution of HMM_MEMORY_RETRIEVE action results in a time charge for the memory search and an update to the HMM_ACTIVE_MEMORY_CELL object which holds the results of the retrieve. If the retrieve was successful the recall_status characteristic of the HMM_ACTIVE_MEMORY_CELL object will have the value RECALLABLE. If the item was not recallable, the value will be

NOT_RECALLABLE. Depending on the case, if an item was successfully recalled, other characteristics of the HMM_ACTIVE_MEMORYCELL object will be updated. For example, for a retrieval of a decimal item, HMM_ACTIVE_MEMORYCELL will contain the following if the recall was successful:

HMM_ACTIVE_MEMORY_CELL

object_name	ON_true_brg_display
char_name	CN_bearing
type_of_item	type_d
value_alpha	blank
value_decimal	78.2
value_whole	99999.0
recall_status	recallable

References

Card, S.K., Moran, T.P., and Newell, A. (1986). The model human processor: An engineering model of human performance. In *Handbook of Perception and Human Performance, Volume II*, K.R. Boff, L. Kaufman, and J.P. Thomas, (Eds.) New York, NY: Wiley-Interscience, 1986.

Cavanaugh, J.P. (1972). Relation between the immediate memory spar and the memory search rate. *Psychological Review*, 79, 525-530.

Posner, M.I. (1978). *Chronometric explorations of the mind.* Hillsdale, NJ: Erlbaum.

SPEAKING

The speaking micromodel determines the time charge for reading aloud based on the vocabulary size and the number of words to be read.

Description

The HOS speaking model determines the time to read a user-specified number of words. The speaking time per word will vary depending on whether the words are elements of a small or a large vocabulary. A small vocabulary is defined as a vocabulary less than 5000 words; a large vocabulary is one with greater than 5000 words. For small vocabularies, the speaking time is 3.4 words per second; for large vocabularies the time is 2.4 words per second. The model is based on data collected by Pierce & Karlin (1957) as reported by McCormick (1970).

HAL Access Statement

The speaking micromodel is invoked as shown below:

```
USING calling_action_object, number_of_words,
    vocabulary_size
    DO HMM_READ_ALOUD
```

where:

calling_action_object is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section).

number_of_words is a whole number variable which holds the number of words to be read.

vocabulary_size is an alphabetic variable with one of two values — LARGE or SMALL.

Speaking Micromodel Example

A sample of a user-defined Action which calls HMM_READ_ALOUD is given below:

```
DEFINITIONS
    OBJECT calling_action object
    CHARACTERISTIC step
    ALPHABETIC vocabulary_size
    WHOLE number_of_words
```

```
ENDDEFINITIONS
    [user code]
SET number_of_words TO [18]
SET vocabulary_size TO [small]
USING calling_action_object, number_of_words,
    vocabulary_size
    DO HMM_READ_ALOUD
```

Execution of the action HMM_READ_ALOUD will result in a simulation time charge which will vary depending on the number of words to be spoken as well as the vocabulary size. The default time charge per word for small and large vocabularies are stored in the words_per_second characteristic of the HMM_VOCABULARY_DATA object set. The following table lists the default settings.

	Set		Default
OBJECT SET	No.	CHARACTERISTIC	
	Value		
HMM_VOCABULARY_DATA			
	1	size_of_vocab	small
		words_per_second	3.4
	2	size_of_vocab	large
		words_per_second	2.4

References

McCormick, E.J. (1970). *Human Factors Engineering, 3rd Edition*. New York, NY: McGraw-Hill Book Co., p.93.

Pierce, J.R., and Karlin, J.E. (1957). Reading rates and the information rate of a human channel. *Bell Telephone Technical Journal*, Vol. 36, pp. 497-516.

VISUAL PERCEPTION MODEL

The visual perception model determines the time required to perceive a visual target.

Description

The default time charge of 0.34 seconds for visual perception is based on data collected by Bois (1971) as reported by Posner (1978). Bois investigated reaction time during a visual matching task, varying the time between stimulus presentation. The default time charge of 0.34 seconds represents the case of no delay between stimuli. This model is appropriate for simulating an operator scanning instruments. Thus this model would be called to simulate the time to perceive the value of a single symbol on a display given the eye has moved to the approximate fixation target in a scan sequence.

HAL Access Statement

The visual perception micromodel is invoked by the following statement:

```
USING calling_action_object
DO HMM_VISUAL_PERCEIVE
```

where:

> **calling_action_object** is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section).

HMM_VISUAL_PERCEIVE determines the time required for perception of a visual target. The value of the time charge is stored in the TIME characteristic of the object HMM_VISUAL_PERCEIVE_STATUS.

Reference

Posner, M.I, (1978). *Chronometric Exploration of the Mind. The Third Paul M. Fitts Lectures.* Hillsdale, NJ: Lawrence Erlbaum Associates Publishers, p.47.

WALKING

The walking micromodel determines the time required for walking per foot of travel.

Description

The default time charge for walking is 190 milliseconds per foot (Clark, 1972).

HAL Access Statement

The walking micromodel is invoked as illustrated below:

USING calling_action_object, feet_to_travel DO HMM_walking

where

calling_action_object is the name of the object associated with the user's action that calls this micromodel (see discussion at the beginning of this section).

feet_to_travel is a decimal variable which defines the number of feet to be traveled.

Walking Micromodel Example

A sample of a user-defined Action which calls HMM_WALKING is given below:

```
DEFINITIONS
    OBJECT calling_action_object
    CHARACTERISTIC step
    DECIMAL feet_to_travel
ENDDEFINITIONS
    [user code]
SET feet_to_travel TO [100.0]
USING calling_action_object, feet_to_travel
     DO HMM_WALKING
```

Execution of the action HMM_WALKING will result in a simulation time charge which will vary depending on the number of feet to be traveled. The default time charge per character is stored in the TIME_PER_FOOT characteristic of the HMM_WALKING_STATUS object.

Reference

Clark, D.O. (1967). The MTM systems of the world. *American Institute of Industrial Engineers Annual Conference Proceedings*, pp. 38.1 -38.10.

HOW TO MODIFY A MICROMODEL

This section describes how to modify the micromodels resident in HOS. Methods are presented for modifying default parameters as well as modifying underlying internal algorithms.

Modifying Default Parameters

Micromodel parameters can be modified by editing micromodel objects directly through the OBJECT EDITOR. The names of the objects associated with each micromodel are listed below:

MICROMODEL	Object Name
HMM_decision	HMM_decision_status
HMM_eye_movement	HMM_eye_movement_status
HMM_fatigue	HMM_fatigue_status
HMM_right_hand_movement	HMM_right_hand_move_status
HMM_left_hand_movement	HMM_left_hand_move_status
HMM_handprint	HMM_handprint_status
HMM_listen	HMM_listen_status
	HMM_listen_error_data (SET of 12)
HMM_right_hand_manip	HMM_control_data (SET of 4)
HMM_left_hand_manip	HMM_control_data (SET of 4)
HMM_memory_store	HMM_memory_data
HMM_memory_retrieve	HMM_memory_data
HMM_read_aloud	HMM_vocabulary_data SET of 2)
HMM_visual_perceive	HMM_visual_percoive_status
HMM_walking	HMM_walking_status

Appendix B contains a list of the objects associated with the micromodels which store timing and error parameters. Default values are also listed.

When modifying a timing parameter, such as the characteristic TIME of the object named HMM_EYE_MOVEMENT_STATUS, the user must enter the new value in seconds. This is necessary since HOS-IV automatically converts all timing parameters associated with micromodels from seconds to the user-specified time unit (for example, converts seconds to tenths of seconds).

Modifying Underlying Algorithms

The HOS micromodels that use only static default parameters in determining simulation timing and errors are:

- HMM_eye_movement,
- HMM_handprint,
- HMM_listen,
- HMM_right_hand_manip (pushbutton, toggle, and rotary dial cases),
- HMM_left_hand_manip (pushbutton, toggle, and rotary dial cases),
- HMM_memory_store
- HMM_read_aloud,
- HMM_visual_perceive, and
- HMM_walking.

To modify these micromodels, the user need only adjust the input parameters as discussed in the preceding section.

All other micromodels, however, are based on specific formulas and algorithms. For example, the hand movement model is based on Fitt's Law and is dependent on the distance the hand must move as well as the width (or positioning accuracy) of the object that the hand must move to. Similarly, the fatigue model is based on an algorithm which adjusts performance times and accuracies dynamically as simulation time elapses.

Micromodels containing underlying algorithms include:

- HMM_decision,
- HMM_fatigue,
- HMM_right_hand_movement,
- HMM_left_hand_movement,
- HMM_right_hand_manip (trackball case).

- HMM_left_hand_manip (trackball case), and
- HMM_memory_retrieve.

Underlying micromodel algorithms are resident in the actions that the user calls to invoke a particular behavioral process. Thus, the right hand movement micromodel is resident in the action HMM_RIGHT_HAND_MOVEMENT. The fatigue micromodel algorithm is resident in HMM_FATIGUE. All micromodel actions are available to the user to view or modify through use of the ACTION EDITOR and are written in the HOS Action Language.

Figure 8-2 is a listing of the right hand movement action named HMM_RIGHT_HAND_MOVEMENT. The underlying algorithm for computing hand movement time is based on Fitt's law:

TIME= k * LOG$_2$ (inches_to_move /
width_in_inches + 0.5)

The user can modify the SET statements in this action to alter the underlying algorithm. Of course, the user must save and translate the newly modified action before creating the simulation which will use the new algorithm.

```
COMMENT                                          HMM_right_hand_movement
     This action is called by the user from any action.
     Three required inputs:        1. object associated with the calling action;
     2. inches_to_move (DECIMAL variable); 3. width_in_inches (DECIMAL
     variable)
ENDCOMMENT
DEFINITIONS
OBJECT     HMM_right_hand move_status, HMM_micromodel_timer,
           HMM_motor_control_output
LOC_OBJECT     calling_action_object
CHARACTERISTIC          state, time, action_time_units, expired_time_units,
                        move_constant, right_hand_movement_time
                   stalemate_flag, micromodel_in_use
DECIMAL    inches_to_move, width_in_inches, k, x, t, a, b ,tsum
WHOLE      clock_ticks
ALPHABETIC      micromodel_name
ENDDEFINITIONS
RECEIVE calling_action_object, inches_to_move, width_in inches
     ENDRECEIVE
SET micromodel_name TO [rhand_move]
IF state OF HMM_right_hand_move_status EQUALS free THEN
     USING micromodel_name DO HMM_motor_controller
     IF stalemate_flag OF HMM_motor_control_output NOT_EQUAL_TO on
     THEN
           .
           .
           GET k FROM move_constant OF HMM_right_hand_move_status
           SET x TO [ (inches_to_move / width_in_inches ) + 0.5]
           SET x TO [log10(x) / log10(2.0)]
           SET t TO [k * x]
           .
           .
END
```

Figure 8-2. Underlying Algorithm: HMM_RIGHT_HAND_MOVEMENT Action

HOW TO BUILD A MICROMODEL

This section describes the general steps necessary in building a new micromodel. At a minimum, the following components are needed:

- An object to store relevant information about the model (for example, input parameters and status information).
- An action which the user will call to initialize the behavior (such as the HMM_RIGHT_HAND_MOVEMENT action presented in Figure 8-2 or the HMM_HANDPRINT action presented in Figure 8-3). This action determines the time required to perform the behavior as well as determining any errors in performance.
- A timer rule which executes the behavioral process (action) over the required number of simulation time intervals.
- An action called by the timer rule to update the time clock for the behavior and to update any other objects and variables as necessary.

The HMM_HANDPRINT model is used as an illustration.

Micromodel Object

An object must be built to store any default parameters associated with the micromodel as well as timing variables which will control the execution of the micromodel during the simulation. The object associated with the HMM_HANDPRINT micromodel is listed below:

OBJECT NAME	CHARACTERISTIC	VALUE
HMM_HANDPRINT_status		
	time_per_character	0.75
	state	free
	action_time_units	0
	expired_time_units	0

```
COMMENT                         HMM_HANDPRINT
    This action is called by the user from any action with two required inputs:
        1. object associated with the calling action;
        2. number of characters to be printed (WHOLE variable)
ENDCOMMENT
DEFINITIONS
OBJECT      HMM_HANDPRINT_status, HMM_motor_control_output
LOC_OBJECT      calling_action_object
CHARACTERISTIC          state, time_per_character, action_time_units,
                        expired_time_units, stalemate_flag,
                micromodel_in_use
DECIMAL     t, tsum
WHOLE       characters, clock_ticks
ALPHABETIC      micromodel_name
ENDDEFINITIONS
RECEIVE calling_action_object, characters  ENDRECEIVE
SET micromodel_name TO [HANDPRINT]
IF state OF HMM_HANDPRINT_status EQUALS free THEN
    USING micromodel_name DO HMM_motor_controller
    IF stalemate_flag OF HMM_motor_control_output NOT_EQUAL_TO on
    THEN
        COMMENT hand is free to print ENDCOMMENT
        PUT busy IN state OF HMM_HANDPRINT_status
        PUT micromodel_name IN micromodel_in_use OF calling_action_object
        GET t FROM time_per_character OF HMM_HANDPRINT_status
        SET t TO [t * characters]
        SET clock_ticks TO [t + 0.5]
        PUT clock_ticks IN action_time_units OF HMM_HANDPRINT_status
        PUT 0 IN expired_time_units OF HMM_HANDPRINT_status
        IF clock_ticks EQUALS 0 THEN
                PUT complete IN state OF HMM_HANDPRINT_status
        ENDIF
    ENDIF
ENDIF
IF state OF HMM_HANDPRINT_status EQUALS complete THEN
    COMMENT printing is complete      ENDCOMMENT
    IF micromodel_in_use OF calling_action_object EQUALS
    micromodel_name
        THEN
        PUT free IN state OF HMM_HANDPRINT_status
        USING calling_action_object DO HMM_update_step
        PUT blank IN micromodel_in_use OF calling_action_object
    ENDIF
ENDIF
END
```

Figure 8-3.   Sample Micromodel Action: HMM_HANDPRINT

The characteristic TIME_PER_CHARACTER stores the default time to HANDPRINT a single character. This parameter will be used in the action HMM_HANDPRINT. The characteristic STATE is also used by HMM_HANDPRINT to determine what processing should occur at each simulation time interval depending on whether the STATE is free, busy, or complete. The characteristics named ACTION_TIME_UNITS and EXPIRED_TIME_UNITS are used by the timer rule to store the number of time intervals required to complete the behavior (ACTION_TIME_UNITS) as well as to track the progress of the behavior (EXPIRED_TIME_UNITS).

Micromodel Action

The micromodel action is called by the user to start a particular behavioral process. In the case of the HOS HANDPRINT model, the user will call the action HMM_HANDPRINT with a USING...DO statement:

```
USING calling_action_object,
   number_of_char  DO HMM_HANDPRINT
```

The HMM_HANDPRINT action, listed in Figure 8-3 receives input variables which specify the user's calling_action_object and the number of characters that must be printed. HMM_HANDPRINT determines how much simulation time will be required to print these characters based on the number of characters and the default time for printing one character (that is, the value stored in the characteristic TIME_PER_CHARACTER of the object HMM_HANDPRINT_STATUS). HMM_HANDPRINT then determines how many simulation time intervals are required to execute this process. As the total time required may be in decimal form, the fraction is rounded to the nearest whole number (clock_ticks). This number then represents how many simulation time intervals must be completed before the handprinting process is finished. This number is placed in the ACTION_TIME_UNITS characteristic of the object HMM_HANDPRINT_STATUS. Also, EXPIRED_TIME_UNITS is initialized to zero.

When initializing an action (that is, when HMM_HANDPRINT is called and the STATE of HMM_HANDPRINT_STATUS is free), the following processing occurs:

- A micromodel controller action named HMM_MOTOR_CONTROLLER is called to determine if the right hand is free to perform handprinting. If the hand is currently moving or manipulating a control, the HMM_HANDPRINT micromodel will be stalemated and the STALEMATE_FLAG characteristic of the object HMM_MOTOR_CONTROL_OUTPUT will be set to on.
- If the hand is free, the STATE of HMM_HANDPRINT_STATUS is set to busy.
- Handprint is placed in the MICROMODEL_IN_USE characteristic of the calling_action_object.
- The time to complete the handprinting process is determined and the appropriate parameters are set (ACTION_TIME_UNITS and EXPIRED_TIME_UNITS) to ensure that the handprint timer rule will fire and thereby execute the process. If the time to complete the process is less than one simulation time interval, the STATE of HMM_HANDPRINT_STATUS is set to complete and the handprint timer rule will not fire.

When handprinting is complete (that is, when HMM_HANDPRINT is called and the STATE of HMM_HANDPRINT_STATUS is complete), the following processing occurs:

- The characteristic STATE of the object HMM_HANDPRINT_STATUS is set to the value FREE.
- The STEP characteristic of calling_action_object is incremented by one by a call to the action HMM_UPDATE_STEP.

- The MICROMODEL_IN_USE characteristic of calling_action_object is set to the value BLANK.

Note: This completion processing only occurs when the action currently calling HMM_HANDPRINT is indeed the action which originally invoked the handprinting process (that is, the MICROMODEL_IN_USE characteristic of the calling_action_object must have the value HANDPRINT). This check is necessary in order to ensure that the STEP characteristic associated with the original calling_action_object is updated.

When handprinting is busy (that is, when HMM_HANDPRINT is called and the STATE of HMM_HANDPRINT_STATUS is busy), no processing occurs.

Micromodel Rule

The micromodel rule is a timer rule which monitors the execution of the process. In the case of the HOS HANDPRINT model, the timer rule states:

```
IF: expired_time_units OF
    HMM_HANDPRINT_status
    LESS_OR_EQUAL action_time_units OF
    HMM_HANDPRINT_status
DO: HMM_HANDPRINT_clock
UNTIL: action_time_units OF
    HMM_HANDPRINT_status EQUALS 0
```

This rule will fire the action HMM_HANDPRINT_clock (described in the following section) if the HANDPRINT processing time is not yet complete.

Micromodel Process Action

The micromodel process action controls the behavioral process that will occur during a single time unit. This action is called by the timer rule associated with a particular micromodel. In the case of the HANDPRINT micromodel, the action HMM_HANDPRINT_CLOCK will update the value of the characteristic EXPIRED_TIME_UNITS of the HMM_HANDPRINT_STATUS object. When EXPIRED_TIME_UNITS is finally updated to equal ACTION_TIME_UNITS, HMM_HANDPRINT_CLOCK will set ACTION_TIME_UNITS to zero. This will have the effect of canceling the micromodel rule by setting the UNTIL condition of the timer rule to true.

Other Considerations

When building a micromodel, the necessary components can, for the most part, be developed by using existing micromodel components as a template. Algorithms, objects, rule, and variables can be modified as appropriate. Another consideration, however, is how the new micromodel will be integrated with the existing micromodels. This consideration will be elaborated in the following section.

MODES OF OPERATION

The preceding sections have described how the micromodels independently work. This section describes how the micromodels are integrated so that particular micromodels can work in parallel. While two cognitive type models may stalemate each other, it is feasible that a cognitive and a psychomotor micromodel can be performed simultaneously. For example, when the simulated operator is performing multiple tasks, a decision can be made on one task while a hand movement is in progress for another task. In the section entitled Parallel Operation, the default settings defining which micromodels can and cannot be run in parallel are presented as well as user procedures to override the default settings.

While it is important to simulate some behaviors in parallel, it is equally important to have the ability to simulate other behaviors in sequence. This is true for the case of an operator performing a specific task which requires that a set of behavioral steps be executed in sequence. This is discussed in the section entitled Sequential Operation.

Parallel Operation

Figure 8-4 illustrates the HOS-IV default settings that control which micromodels can and cannot be executed in parallel. An X marked in any column indicates the micromodels which cannot be run in parallel. For example, when an action calls the eye movement micromodel, if the eye movement model itself or the visual perception micromodel are busy, a new eye movement will not be executed. However, if these two micromodels are free, an eye movement can execute in parallel with other cognitive or motor behaviors. In general, the default structure allows simultaneous execution of a single micromodel from major behavioral groupings — that is, perceptual, cognitive, and motor. Micromodels within a particular grouping will in general compete with each other and thereby stalemate performance.

The micromodel stalemate structure has been flexibly designed so that the user can easily redefine which micromodels will compete with each other. Redefinition can be accomplished by modifying two major object sets resident in the Object Editor — HMM_cognitive_control and HMM_motor_controller. The following table lists the default settings for one of these object sets:

OBJECT SET		CHARACTERISTIC	Default
HMM_COGNITIVE_CONTROL			
	1	micromodel_name	eye_move
		eye_move_compete	on
		visual_perceive_compete	on
		decision_compete	off
		memory_store_compete	off
		memory_retrieve_compete	off
		read_aloud_compete	off
		listen_compete	off
	2	micromodel_name	
	visual_perceive		
		eye_move_compete	on
		visual_perceive_compete	on
		decision_compete	on
		memory_store_compete	on
		memory_retrieve_compete	on

MICROMODEL STALEMATE SCHEDULE	Eye Movement	Visual Perception	Memory Store	Memory Retrieve	Decision	Listening	Speaking	Hand Printing	Rt Hand Movement	Left Hand Movement	Rt Hand Manip	Left Hand Manip	Walking
Eye Movement	X	X											
Visual Perception	X	X	X	X	X								
Memory Store		X	X	X	X	X	X						
Memory Retrieve		X	X	X	X	X	X						
Decision		X	X	X	X	X	X						
Listening			X	X	X	X	X						
Speaking			X	X	X	X	X						
Hand Printing								X	X		X		X
Rt Hand Movement								X	X		X		
Left Hand Movement										X		X	
Rt Hand Manip								X	X		X		X
Left Hand Manip										X		X	X
Walking								X			X	X	X

X indicates these micromodels cannot be run in parallel

Figuer 8-4. Default Settings for Parallel Micromodel Operation

	read_aloud_compete	off
	listen_compete	off
3	micromodel_name	decision
	eye_move_compete	off
	visual_perceive_compete	on
	decision_compete	on
	memory_store_compete	on
	memory_retrieve_compete	on
	read_aloud_compete	on
	listen_compete	on
4	micromodel_name	memory_store
	eye_move_compete	off
	visual_perceive_compete	on
	decision_compete	on
	memory_store_compete	on
	memory_retrieve_compete	on
	read_aloud_compete	on
	listen_compete	on
5	micromodel_name	memory_retrieve
	eye_move_compete	off
	visual_perceive_compete	on
	decision_compete	on
	memory_store_compete	on
	memory_retrieve_compete	on
	read_aloud_compete	on
	listen_compete	on
6	micromodel_name	read_aloud
	eye_move_compete	off
	visual_perceive_compete	off
	decision_compete	on
	memory_store_compete	on
	memory_retrieve_compete	on
	read_aloud_compete	on
	listen_compete	on
7	micromodel_name	listen
	eye_move_compete	off
	visual_perceive_compete	off
	decision_compete	on
	memory_store_compete	on
	memory_retrieve_compete	on
	read_aloud_compete	on
	listen_compete	on

To alter the default structure on parallel operations, the user can modify the appropriate items within the object set. For example, to allow simultaneous execution of the visual perception model with the decision model, the user would open set item two — HMM_cognitive_control002 — and change the value of decision_compete to off. Similarly, the user would also open set item three — HMM_cognitive_control003 —and change the value of visual_perceive_compete to off.

Sequential Operation

While HOS-IV allows simultaneous performance of some micromodels, a method for executing behaviors in a predefined sequence is also important. This is necessary for developing operator actions which contain a list of the sequential steps or behaviors required to accomplish a specific mission task or subtask. To achieve this goal, the user must separate the sequential behaviors within an operator action in such a way that the next step will only be performed after all previous steps have been completed. To assist the user in this process, the following HOS micromodel structure has been developed:

- All HOS micromodels (with the exception of HMM_FATIGUE) accept a user-defined object that is associated with the calling action. The user must create this object with two characteristics:

  (1) STEP with an initial value of 1, and;

  (2) MICROMODEL_IN_USE with an initial value of blank.
- At the completion of each micromodel behavior, the micromodel will increment the value of the characteristic STEP by one. This will have the affect of allowing the next step in the user-defined action to occur.

A sample user action which accesses HOS micromodels in this manner is illustrated in Figure 8-1.

9. CREATING AND RUNNING A SIMULATION

This section describes how to create a simulation once all the components — objects, events, rules, and actions — have been defined and how to run a simulation.

CREATE SIMULATION

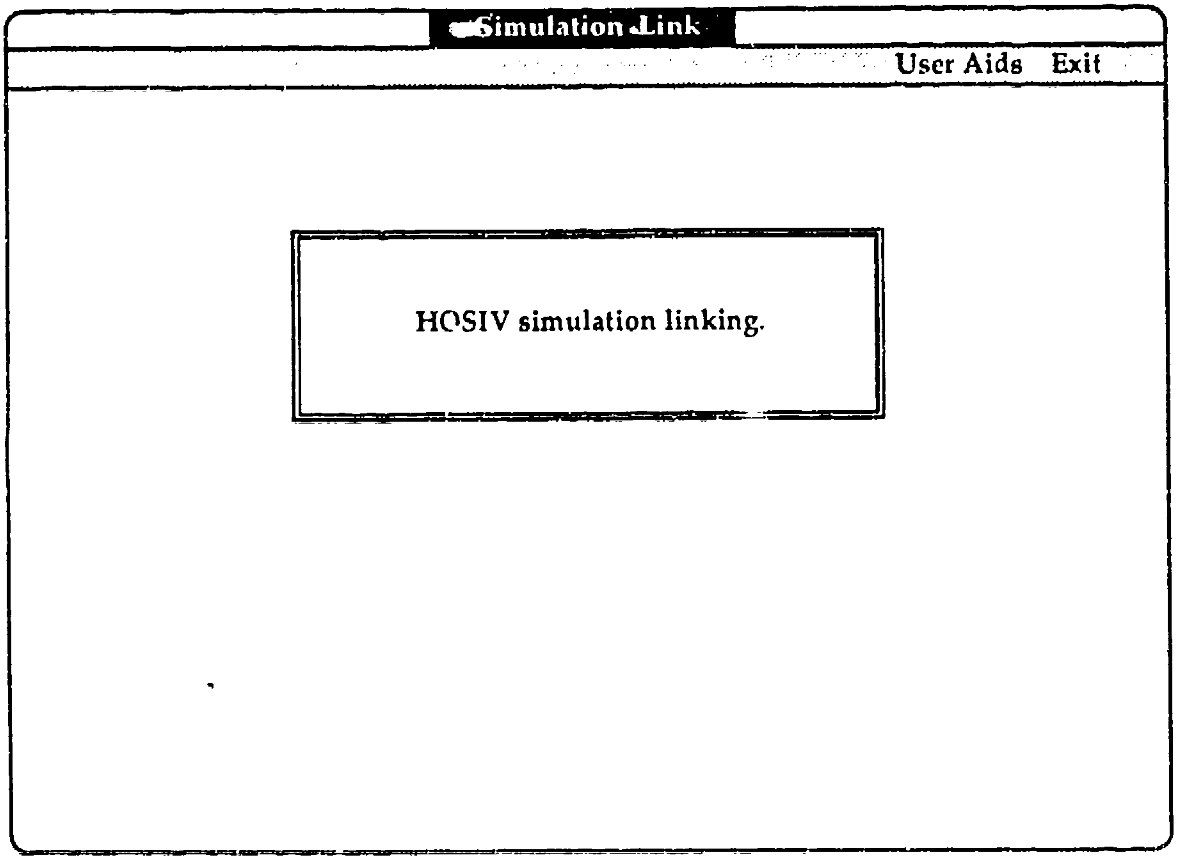
Create Simulation builds the HOS simulation based upo: the defined events, rules, objects, and actions. It reviews all the individual definitions to ensure that each referenced item has been defined. These checks include:

- Ensuring that all actions referenced in events have been defined,
- Ensuring that all actions, alphabetics, and object-characteristic pairs referenced in rules have been defined, and
- Ensuring that all object-characteristic pairs, alphabetics, and actions referenced in actions have been defined.

If all cross-references have been validated, then the simulation is created.

Informative message windows, as illustrated below, are displayed to inform the user of the status of the simulation creation.

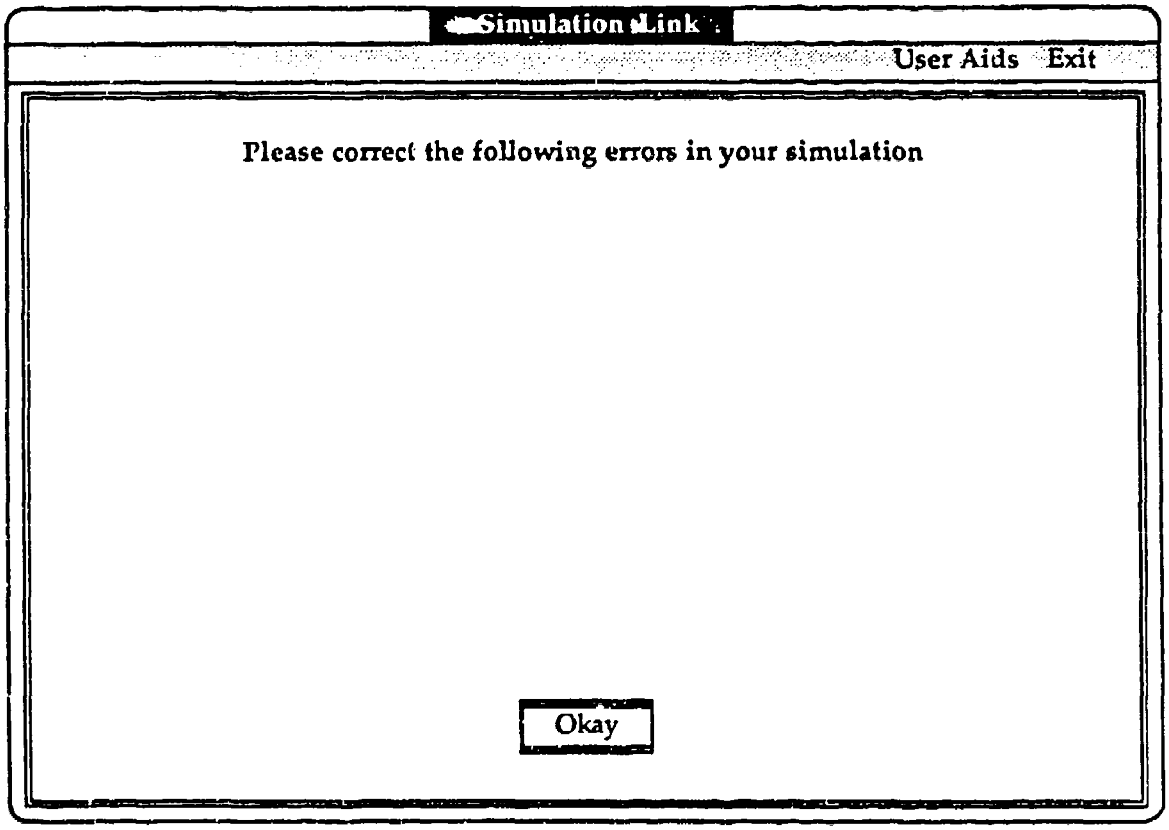
Create Simulation Information Window



HOSIV simulation linking.

If any errors are detected, an error message screen, as illustrated below, is displayed. This window contains a pushbutton for the user to depress after the error message has been comprehended in order to continue.

Create Simulation Error Messages



Please correct the following errors in your simulation

The undefined items must be specified using the appropriate HOS editor and the Create Simulation step repeated until all items have been defined. For example, if the message 'object_a is undefined' is generated, the user would enter the Object Editor to define object_a. This message will also occur if the name of an item is misspelled, e.g., objec_a should be object_a.

RUN SIMULATION

Run Simulation executes the selected simulation and generates the simulation output files. It creates a screen showing the simulation status at each time increment and the currently active event, rule, and action. Once the simulation has been created, the Run Simulation step can be repeated as often as necessary. In addition, the values of object characteristics can be changed using the Object Editor and the simulation rerun without having to repeat the Create Simulation step. Similarly, the time of an event can also be changed without recreating the simulation.

Starting the Simulation

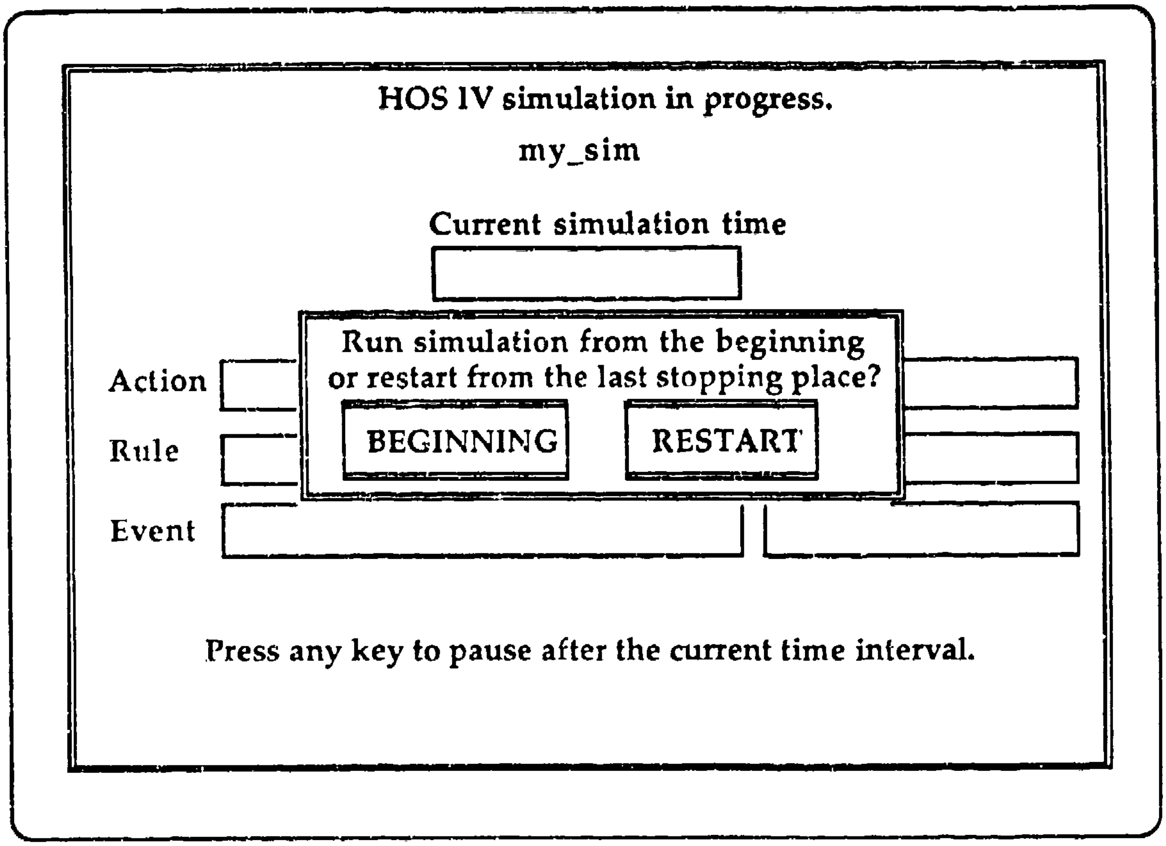
The screen shown below allows the user to indicate whether the simulation is to start at the beginning or to be restarted at the point where it was previously terminated. The selection of the **BEGINNING** pushbutton indicates that the simulation is to start at the start time specified in Simulation Setup. The use of the **RESTART** pushbutton indicates that the simulation is to start where it was previously terminated. If either the event times or object values are changed, the simulation can only be restarted from the beginning.

Run Simulation contains an option that permits the simulation to be interrupted at any point When interrupted, two options are available:

1. Continue the simulation, or
2. Stop executing the simulation but create the necessary files so that the simulation can be restarted later.

If the second option is selected, the simulation results are available for analysis through View Results (see Section 10). These results will contain information from the start of the simulation (whether the beginning time or the restart time) to the time of interruption. However, if the simulation is restarted at a later time, the results of that run will only contain information from the restarted time.

Start Simulation Screen



HOS IV simulation in progress.

my_sim

Run simulation from the beginning or restart from the last stopping place?

Press any key to pause after the current time interval.

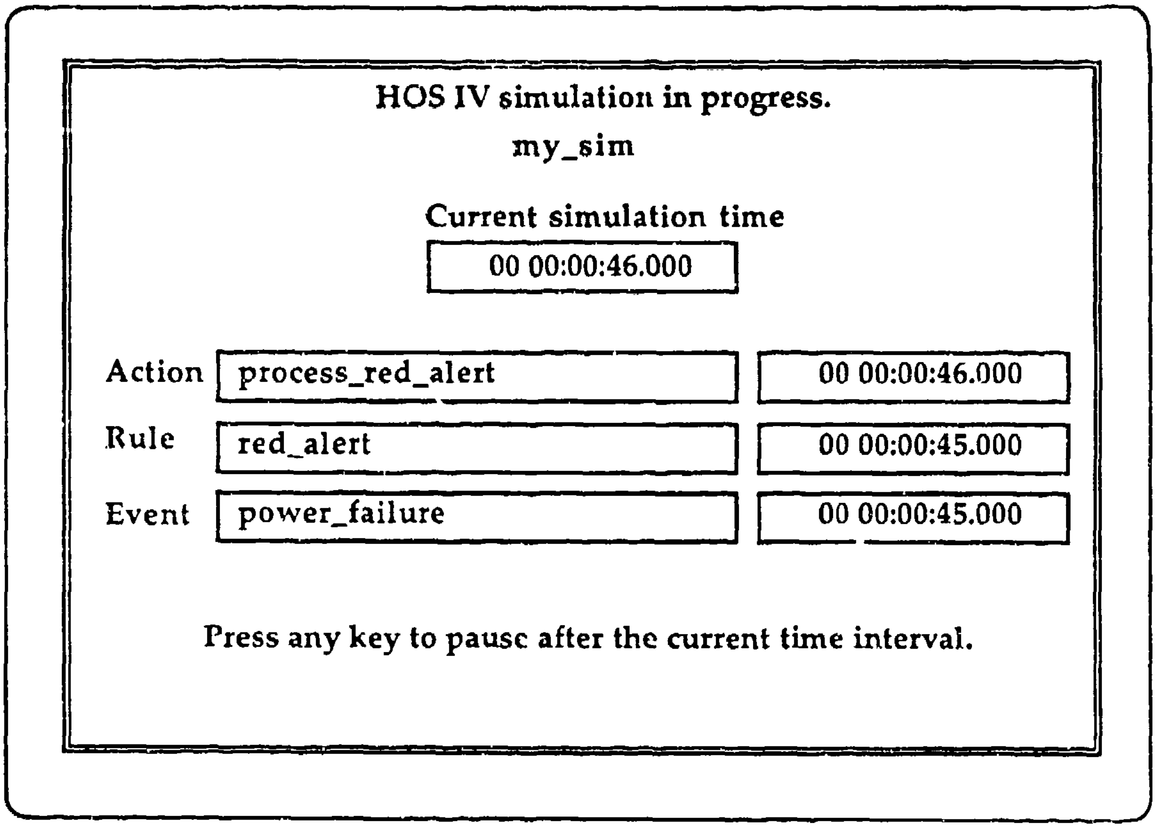
Running the Simulation

The run simulation screen, shown below, shows the current status of the simulation through a series of labeled boxes as follows:

- simname — the name of currently executing simulation. In this example, the simulation name is my_sim.

- Current simulation time — the time of the current simulation in the form dd hh:mm:ss.ttt where dd is days, hh is hours, mm is minutes, ss is seconds and ttt is thousands of seconds. In the example below, the current time is 46 seconds.
- Action — the name of the action currently being processed by the simulation and the time the action started. In the example below, the action is process_red_alert at time 46 seconds.
- Rule — the name of the rule currently invoked and the time the rule was triggered. In the example below, the rule is red_alert at time 45 seconds.
- Event — the name of the current event and the time the event started. In the example below, the event is power_failure at time 45 seconds.

Run Simulation Screen



HOS IV simulation in progress.

my_sim

Current simulation time

00 00:00:46.000

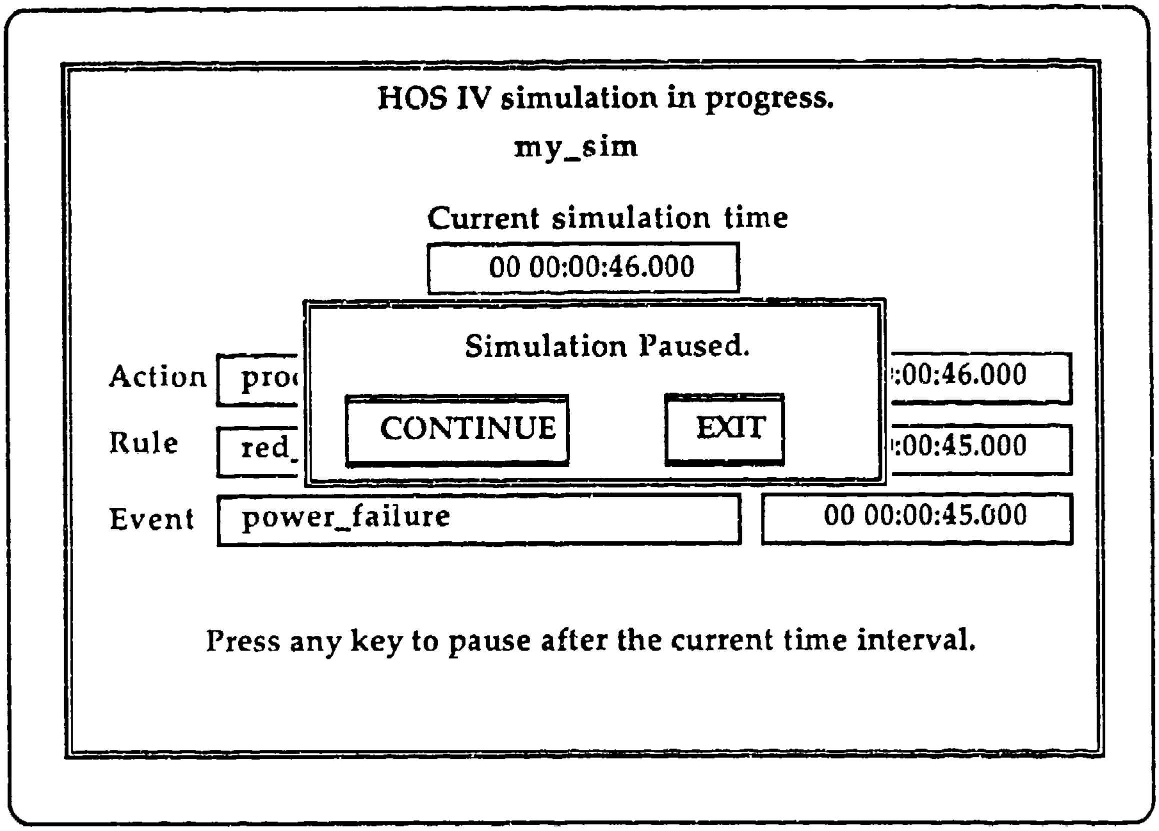
Action process_red_alert 00 00:00:46.000

Rule red_alert 00 00:00:45.000

Event power_failure 00 00:00:45.000

Press any key to pause after the current time interval.

Simulation Pause Screen



HOS IV simulation in progress.

my_sim

Current simulation time

00 00:00:46.000

Simulation Paused.

Action pro :00:46.000

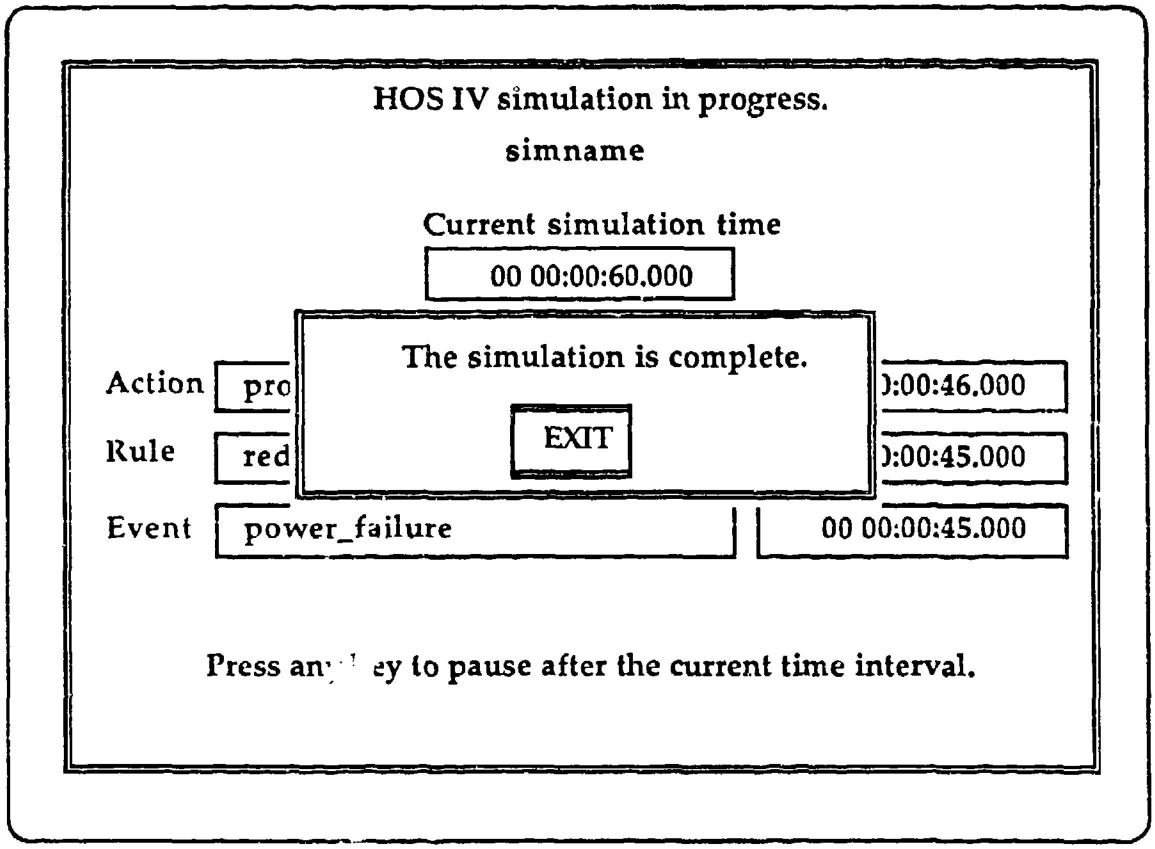
Rule red_ :00:45.000

Event power_failure 00 00:00:45.000

Press any key to pause after the current time interval.

When the normal end of the simulation is reached, the screen displayed below will be displayed.

End of Simulation Screen



HOS IV simulation in progress.

simname

Current simulation time

00 00:00:60.000

The simulation is complete.

Action pro):00:46.000

Rule red):00:45.000

Event power_failure 00 00:00:45.000

Press any key to pause after the current time interval.

10. VIEWING THE RESULTS

The view results module is used to examine data produced by a simulation. The View Results module contains a series of standard reports that are used to assist in the analysis of simulation results. All reports contain a header displaying the simulation name, start time, description, run date and run time.

REPORTS

The available reports are described below. All reports are presented in standard time units dd:hh:mm:ss.ttt:

where

dd is days,

hh is hours,

mm is minutes,

ss is seconds,

ttt is thousandths of a second.

Object Analysis

The object analysis report traces the value of the characteristics of a selected object during the simulation and displays the time and object values whenever the value of the object characteristic changes. A screen will be displayed that contains a list selection box to be used to select an object for analysis. The following information will be displayed for the selected object:

Time	Object Name	Characteristic	Previous Value	Current Value
10:10.000	Symbol	Color	Green	Yellow
20:24.000	Symbol	Size	8	10
20:24.000	Symbol	Color	Yellow	Red

This example shows that the color of the symbol changed from green to yellow at 10:10.000 and from yellow to red at time 20:24.000. Also at time 20:24.000, the size of the symbol changed. This report is useful to track the contents of selected objects.

Rule Analysis

The rule analysis report generates statistics on rule usage including the number of times the rule was active, the total time the rule was active, the mean and standard deviation of the rule active time, and the percent of the simulation time that the rule was active. This information is also collected for the number of times the rule was interrupted. A rule is defined as active during the time interval when the IF component of a rule is true and the UNTIL component is false. A rule is defined as interrupted when it is deactivated through use of a SUSPEND verb or when it was deactivated due to simulation termination.

The rule analysis report is illustrated below:

```
Operator Rules

Rule 400 (verbal_contact_report)

                        Total                          %Sim
          Count         Time        Mean     Std Dev   Time

Active     20        40:20.000   2:17.500  20.000      53
Interrupts 5         09:10.100   1:50.300  30.000       9
```

The above data for operator rule 400 — verbal_contact_report — indicates that the rule was active 20 times and interrupted 5 times during the simulation. The rule was active a total time of 40 minutes and 20 seconds. For interrupt cases, the rule had been active a total of 9 minutes and 10.1 seconds before being suspended. The mean active time for the rule was 2 minutes and 17.5 seconds with a standard deviation of 20 seconds. The mean time for the interrupted cases was approximately 1 minute and 50 seconds indicating that the task was usually almost complete when the interrupts occurred. This rule was active 53 percent of the simulation time.

User Defined Simulation Output

The user-defined report are those produced through use of FILE verbs contained in actions. The report will contain the standard heading and the information however specified within the actions.

Action Timeline

The action timeline generates a timeline showing the name of each active action at each time interval in the simulation as shown below:

Time	Action Name
00:00:00.000	HMM_use_micromodels
00:00:00.000	SAM_start_sampler
00:00:00.100	SAM_main_monitor
00:00:00.100	SAM_secondary_monitor

Event Timeline

The event timeline generates a timeline showing the time and name that each event occurred. The report is illustrated below:

Time	Event Name
00:00:00.000	Setup_tests
00:00:01.000	Start_test1
00:00:10.000	Start_test2

Object Timeline

The object timeline generates a timeline which indicates when each characteristic of an object was modified as shown below:

Time	Object Name	Characteristic	Previous Value	Current Value
00:00:00.100	symbol	Color	Green	Yellow
00:00:00.200	right_hand	status	resting	moving
00:00:00.200	symbol	Color	Yellow	Red
00:00:00.400	right_hand	x_location	10.2	10.3
00:00:00.400	right_hand	y_location	41.5	41.4

Rule Timeline

The rule timeline report generates a status timeline for all active rule at each simulation time interval.

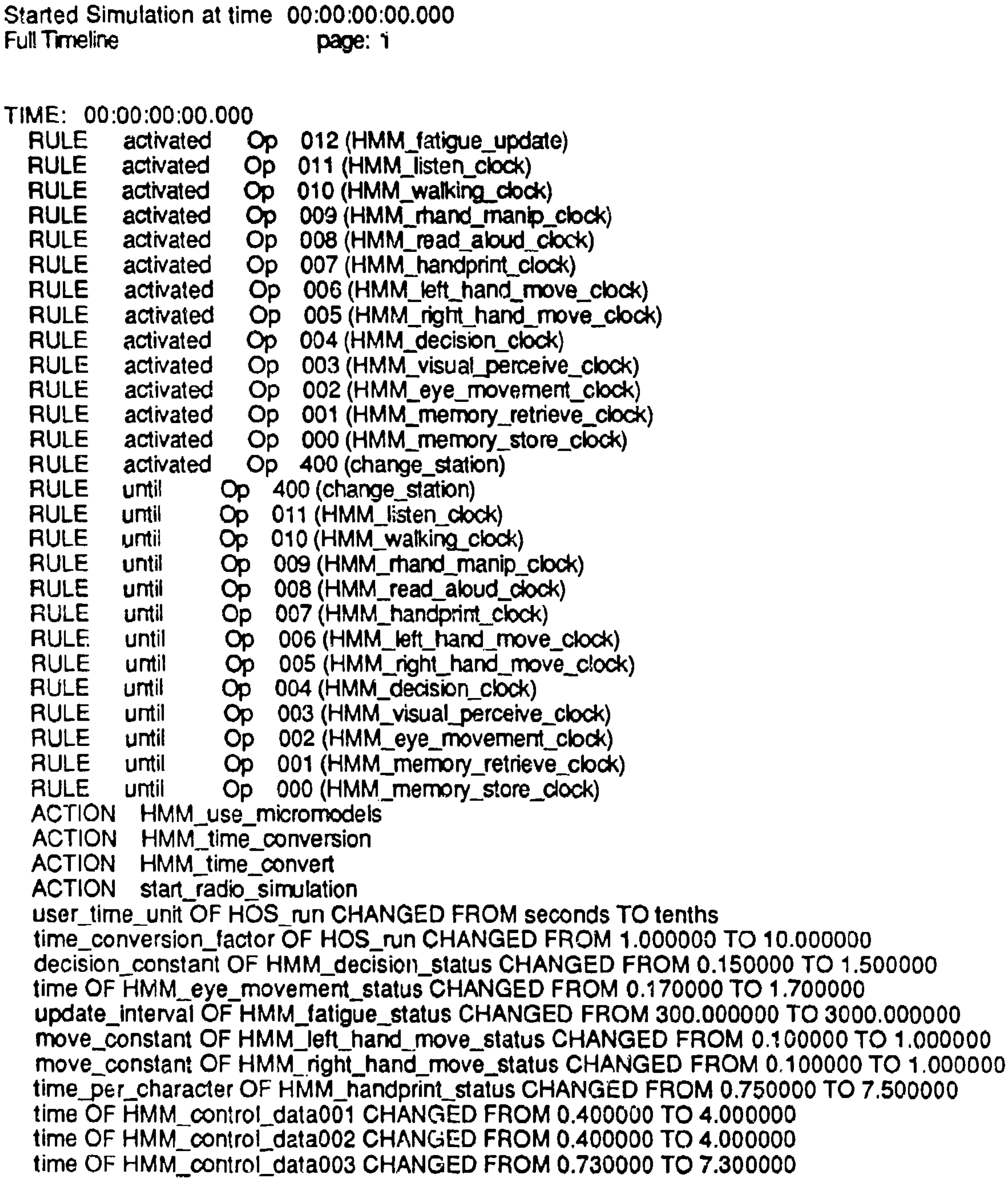
```
Time          Status       Rule Number    Rule Name

00:00:02.000  activated    800            Setup_tests
00:00:02.000  if true      801            Start_test1
00:00:03.000  until true   802            Start_test2
00:00:03.000  suspended    801            Start_test1
```

An entry to this report is generated whenever:

- A rule is added to the active rule list (activated),
- The IF clause of a rule is evaluated and the condition is true (if true),
- The UNTIL clause of a rule is evaluated and the condition is true (until true), and
- A rule is removed from the active rule list (suspended).

Full Timeline

The full timeline report combines the event, rule, action, and object timeline into one single report and shows what is happening during each simulation time interval as shown in Figure 10-1. At time zero, or at the simulation start time, if the INCLUDE MICROMODELS option is selected from the Setup Simulation Editor, the output will list the micromodel parameter and rule initialization. In Figure 10-1, no operator activity is reported until time 00:00:00:01.000 when an event — Turn to WIOQ — happened. At this time, HOS is simulating someone listening to a command to tune the radio to station WIOQ.



```
Started Simulation at time  00:00:00:00.000
Full Timeline                    page: 1


TIME:  00:00:00:00.000
   RULE    activated    Op   012 (HMM_fatigue_update)
   RULE    activated    Op   011 (HMM_listen_clock)
   RULE    activated    Op   010 (HMM_walking_clock)
   RULE    activated    Op   009 (HMM_rhand_manip_clock)
   RULE    activated    Op   008 (HMM_read_aloud_clock)
   RULE    activated    Op   007 (HMM_handprint_clock)
   RULE    activated    Op   006 (HMM_left_hand_move_clock)
   RULE    activated    Op   005 (HMM_right_hand_move_clock)
   RULE    activated    Op   004 (HMM_decision_clock)
   RULE    activated    Op   003 (HMM_visual_perceive_clock)
   RULE    activated    Op   002 (HMM_eye_movement_clock)
   RULE    activated    Op   001 (HMM_memory_retrieve_clock)
   RULE    activated    Op   000 (HMM_memory_store_clock)
   RULE    activated    Op   400 (change_station)
   RULE    until     Op   400 (change_station)
   RULE    until     Op   011 (HMM_listen_clock)
   RULE    until     Op   010 (HMM_walking_clock)
   RULE    until     Op   009 (HMM_rhand_manip_clock)
   RULE    until     Op   008 (HMM_read_aloud_clock)
   RULE    until     Op   007 (HMM_handprint_clock)
   RULE    until     Op   006 (HMM_left_hand_move_clock)
   RULE    until     Op   005 (HMM_right_hand_move_clock)
   RULE    until     Op   004 (HMM_decision_clock)
   RULE    until     Op   003 (HMM_visual_perceive_clock)
   RULE    until     Op   002 (HMM_eye_movement_clock)
   RULE    until     Op   001 (HMM_memory_retrieve_clock)
   RULE    until     Op   000 (HMM_memory_store_clock)
   ACTION   HMM_use_micromodels
   ACTION   HMM_time_conversion
   ACTION   HMM_time_convert
   ACTION   start_radio_simulation
   user_time_unit OF HOS_run CHANGED FROM seconds TO tenths
   time_conversion_factor OF HOS_run CHANGED FROM 1.000000 TO 10.000000
   decision_constant OF HMM_decision_status CHANGED FROM 0.150000 TO 1.500000
   time OF HMM_eye_movement_status CHANGED FROM 0.170000 TO 1.700000
   update_interval OF HMM_fatigue_status CHANGED FROM 300.000000 TO 3000.000000
   move_constant OF HMM_left_hand_move_status CHANGED FROM 0.100000 TO 1.000000
   move_constant OF HMM_right_hand_move_status CHANGED FROM 0.100000 TO 1.000000
   time_per_character OF HMM_handprint_status CHANGED FROM 0.750000 TO 7.500000
   time OF HMM_control_data001 CHANGED FROM 0.400000 TO 4.000000
   time OF HMM_control_data002 CHANGED FROM 0.400000 TO 4.000000
   time OF HMM_control_data003 CHANGED FROM 0.730000 TO 7.300000
```

Figure 10-1. Sample Full Timeline Report

```
    time OF HMM_control_data004 CHANGED FROM 0.100000 TO 1.000000
    retr_label_time OF HMM_memory_data CHANGED FROM 0.070000 TO 0.700000
    search_rate_per_item OF HMM_memory_data CHANGED FROM 0.047000 TO 0.470000
    storage_time OF HMM_memory_data CHANGED FROM 0.100000 TO 1.000000
    half_life OF HMM_memory_data CHANGED FROM 7.000000 TO 70.000000
    time OF HMM_visual_perceive_status CHANGED FROM 0.340000 TO 3.400000
    time_per_foot OF HMM_walking_status CHANGED FROM 0.190000 TO 1.900000

TIME:  00:00:00:00.100
    RULE     until      Op   400 (change_station)
    RULE     until      Op   011 (HMM_listen_clock)
    RULE     until      Op   010 (HMM_walking_clock)
    RULE     until      Op   009 (HMM_rhand_manip_clock)
    RULE     until      Op   008 (HMM_read_aloud_clock)
    RULE     until      Op   007 (HMM_handprint_clock)
    RULE     until      Op   006 (HMM_left_hand_move_clock)
    RULE     until      Op   005 (HMM_right_hand_move_clock)
    RULE     until      Op   004 (HMM_decision_clock)
    RULE     until      Op   003 (HMM_visual_perceive_clock)
    RULE     until      Op   002 (HMM_eye_movement_clock)
    RULE     until      Op   001 (HMM_memory_retrieve_clock)
    RULE     until      Op   000 (HMM_memory_store_clock)

TIME:  00:00:00:00.200
    RULE     until      Op   400 (change_station)
    RULE     until      Op   011 (HMM_listen_clock)
    RULE     until      Op   010 (HMM_walking_clock)
    RULE     until      Op   009 (HMM_rhand_manip_clock)
    RULE     until      Op   008 (HMM_read_aloud_clock)
    RULE     until      Op   007 (HMM_handprint_clock)
    RULE     until      Op   006 (HMM_left_hand_move_clock)
    RULE     until      Op   005 (HMM_right_hand_move_clock)
    RULE     until      Op   004 (HMM_decision_clock)
    RULE     until      Op   003 (HMM_visual_perceive_clock)
    RULE     until      Op   002 (HMM_eye_movement_clock)
    RULE     until      Op   001 (HMM_memory_retrieve_clock)
    RULE     until      Op   000 (HMM_memory_store_clock)

.
.
.
.
.
.
.
```

Figure 10-1. Sample Full Timeline Report (Continued)

```
TIME:  00:00:00:01.000
  EVENT       Event turn to WIOQ happened
  RULE    if          Op   400 (change_station)
  RULE    activated   Op   400 (change_station)
  RULE    if          Op   011 (HMM_listen_clock)
  RULE    activated   Op   011 (HMM_listen_clock)
  RULE    until       Op   010 (HMM_walking_clock)
  RULE    until       Op   009 (HMM_rhand_manip_clock)
  RULE    until       Op   008 (HMM_read_aloud_clock)
  RULE    until       Op   007 (HMM_handprint_clock)
  RULE    until       Op   006 (HMM_left_hand_move_clock)
  RULE    until       Op   005 (HMM_right_hand_move_clock)
  RULE    until       Op   004 (HMM_decision_clock)
  RULE    until       Op   003 (HMM_visual_perceive_clock)
  RULE    until       Op   002 (HMM_eye_movement_clock)
  RULE    until       Op   001 (HMM_memory_retrieve_clock)
  RULE    until       Op   000 (HMM_memory_store_clock)
  ACTION   request_station_change
  ACTION   change_station
  ACTION   HMM_listen
  ACTION   HMM_cognitive_controller
  ACTION   HMM_listen_clock
  ACTION   HMM_update_clock
  desired_station OF radio CHANGED FROM WMMR TO WIOQ
  stalemate_flag OF HMM_cognitive_contrl_output CHANGED FROM off TO off
  state OF HMM_listen_status CHANGED FROM free TO busy
  micromodel_in_use OF change_station_status CHANGED FROM blank TO listen
  action_time_units OF HMM_listen_status CHANGED FROM 0 TO 25
  expired_time_units OF HMM_listen_status CHANGED FROM 0 TO 0
  listening_time OF HMM_micromodel_timer CHANGED FROM 0.000000 TO 25.000000
  expired_time_units OF HMM_listen_status CHANGED FROM 0 TO 1

TIME:  00:00:00:01.100
  RULE    if          Op   400 (change_station)
  RULE    activated   Op   400 (change_station)
  RULE    if          Op   011 (HMM_listen_clock)
  RULE    activated   Op   011 (HMM_listen_clock)
  RULE    until       Op   010 (HMM_walking_clock)
  RULE    until       Op   009 (HMM_rhand_manip_clock)
  RULE    until       Op   008 (HMM_read_aloud_clock)
  RULE    until       Op   007 (HMM_handprint_clock)
  RULE    until       Op   006 (HMM_left_hand_move_clock)
  RULE    until       Op   005 (HMM_right_hand_move_clock)
  RULE    until       Op   004 (HMM_decision_clock)
  RULE    until       Op   003 (HMM_visual_perceive_clock)
```

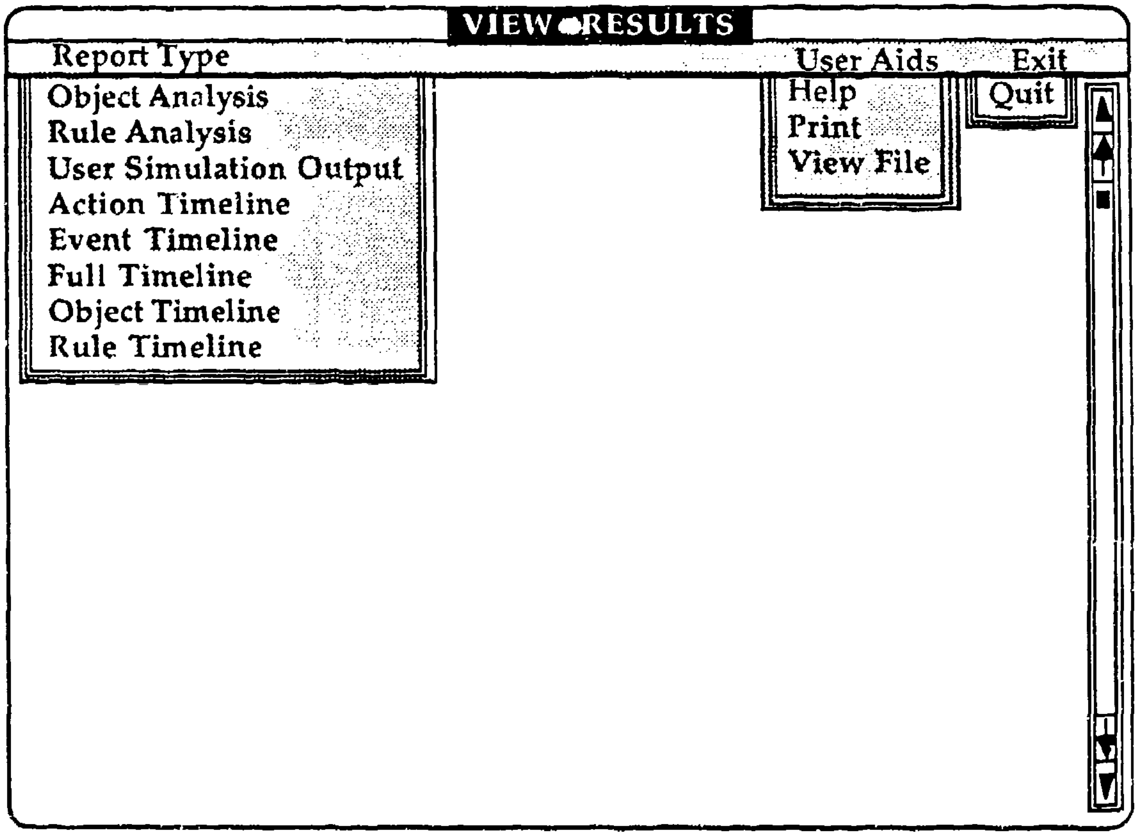
Figure 10-1. Sample Full Timeline Report (Continued)

```
   RULE    until      Op   002 (HMM_eye_movement_clock)
   RULE    until      Op   001 (HMM_memory_retrieve_clock)
   RULE    until      Op   000 (HMM_memory_store_clock)
   ACTION   change_station
   ACTION   HMM_listen
   ACTION   HMM_listen_clock
   ACTION   HMM_update_clock
   expired_time_units OF HMM_listen_status CHANGED FROM 1 TO 2

TIME:  00:00:00.01.200
   EVENT      OUT OF EVENTS
   RULE    if         Op   400 (change_station)
   RULE     activated  Op   400 (change_station)
   RULE    if         Op   011 (HMM_listen_clock)
   RULE     activated  Op   011 (HMM_listen_clock)
   RULE    until      Op   010 (HMM_walking_clock)
   RULE    until      Op   009 (HMM_rhand_manip_clock)
   RULE    until      Op   008 (HMM_read_aloud_clock)
   RULE    until      Op   007 (HMM_handprint_clock)
   RULE    until      Op   006 (HMM_left_hand_move_clock)
   RULE    until      Op   005 (HMM_right_hand_move_clock)
   RULE    until      Op   004 (HMM_decision_clock)
   RULE    until      Op   003 (HMM_visual_perceive_clock)
   RULE    until      Op   002 (HMM_eye_movement_clock)
   RULE    until      Op   001 (HMM_memory_retrieve_clock)
   RULE    until      Op   000 (HMM_memory_store_clock)
   ACTION   change_station
   ACTION   HMM_listen
   ACTION   HMM_listen_clock
   ACTION   HMM_update_clock
   expired_time_units OF HMM_listen_status CHANGED FROM 2 TO 3
```

Figure 10-1. Sample Full Timeline Report (Continued)

VIEW RESULTS SCREENS

The View Results module consists of a title bar, a menu bar, a text viewing window with a scrollbar, and a number of dialog boxes used for program interaction with the user as illustrated below.



View Results Title Bar

The title bar contains the words 'View Results' as the function name in the center. It does not use the current activity or status information areas of the title bar.

View Results Menu Bar

The menu bar for the View Results module contains the following menu options:

- **Report Type** — file related commands such as saving, opening, etc.
- **User Aids** — provides the capabilities to view help files and action files.
- **Exit** — terminates View results module and returns the user to the HOSIV module.

Report Type Commands

The **Report Type** pull down menu contains:

- **Object Analysis** — opens the Object Analysis report.
- **Rule Analysis** — opens the Rule Analysis report.
- **User Simulation Output** — opens the User Simulation Output report.
- **Action Timeline** — opens the Action Timeline report.
- **Event Timeline** — opens the Event Timeline report.
- **Full Timeline** — opens the Full Timeline report.
- **Object Timeline** — opens the Object Timeline report.
- **Rule Timeline** — opens the Rule Timeline report.

User Aids Commands

The **User Aids** pull down menu contains commands that allow the user receive help on the current module and view action files.

- **Help** — allows the user to obtain additional information about using the View Results module.

- **View File** — allows the user to view action files created using the view results module.

View Results Text Viewing Window

The text viewing window consists of text area and a scrollbar and is illustrated below. The text area is 23 rows by 77 columns. The scrollbar is to the right of the text area and has four control buttons and a relative file position indicator (■). The four controls on the scrollbar are:

- **Scroll Line Down (▲)**— displays a page of text starting from the line before the current top line.
- **Scroll Page Down (↑)**— displays a page of text starting one page before the current top line.
- **Scroll Page Up (↓)**— displays a page of text starting from the current bottom line.
- **Scroll Line Up (▼)**— displays a page of text starting from the line after the current top line.

View Results Dialog Boxes

If the selected report requires additional information, text entry boxes are generated to request the appropriate information.

View Results Message Windows

Message windows display information which the user must acknowledge by depressing one of the selected pushbutton.

- **End viewing session** — confirmation that the view results function is terminating.

View Results Information Windows

The following informative messages indicating that the system is carrying out the entered command and to let the user know the status of the operation are contained in View Results.

- **Reading file** — the file is being read.

11. SAMPLE SIMULATIONS

This section describes two sample HOS simulations:

1. RADIO — simulates someone performing a sequential set of steps to change the station of a car radio, and
2. SAMPLER — simulates an operator monitoring two displays in parallel and responding to an emergency alarm.

Both of these simulations use the HOS micromodel library described in Section 8. These simulations are standardly included in the HOS software system and are directly viewable through the various editors.

GENERAL STEPS IN BUILDING SIMULATIONS

The general steps to follow when developing a simulation are:

1. Analyze the system to be simulated, isolating independent tasks and functions.
   - Write ACTIONS which describe the processing which must occur in order to accomplish each task.
   - Determine measures of effectiveness to be tracked during the simulation.
   - Build an object data base to support these ACTIONS and measures of effectiveness.
2. Determine what ACTIONS must occur at every simulation time interval. Write a rule for each of these ACTIONS. For example,

   IF: status OF HOS_run EQUALS on
   DO: SAM_main_monitor
   UNTIL: status OF HOS_run EQUALS off

   This rule will invoke the ACTION named SAM_main_monitor at every simulation time interval that the above rule is active (that is, has been activated by a START verb).

3. Determine what ACTIONS will occur when specific simulation conditions are met. Write a rule which conditionally calls those ACTIONS. For example,

   IF: current_station OF radio NOT_EQUAL_TO desired_station OF radio
   DO: change_station
   UNTIL: current_station OF radio EQUALS desired_station OF radio

   When the above rule is active, the ACTION named change_station will be invoked only at simulation time intervals that the current station of the radio is not equal to the desired station of the radio.

4. Determine what events will occur in the simulation.
   - Use the Event Editor to define the time of the event as well as the ACTION called by the event.
   - Write the ACTION called by the event. This action sets the conditions that will alter the simulation flow (for example, START or SUSPEND rules or modify the characteristics of objects).
   - Build an object data base to support these event-related ACTIONS as necessary.
5. Define the simulation startup action.
   - Use the Simulation Setup Editor to define simulation parameters.
   - Write the startup simulation ACTION which activates the appropriate rules at the beginning of the simulation.
   - Build the object data base to support this ACTION as needed.
6. Create the simulation.

7. Run the simulation.
8. View the results.

If additional information is desired, other than what is provided in the standard output files of the ViewResults Editor, it is recommended that the user employ FILE and PRINT statements within the ACTIONS. These statements are particularly useful for debugging and for printing measures of effectiveness.

RADIO SIMULATION

RADIO simulates an automobile driver listening to radio station WMMR when a passenger requests a station change to listen to WIOQ. The driver observes that the current station is not WIOQ and determines which radio button to depress to tune to that station. The general steps outlined above for building simulations are used to illustrate the development of the RADIO simulation.

For Step 1, the CHANGE_STATION ACTION was defined. CHANGE_STATION contains the behavioral steps associated with changing the radio station:

- Listen to what station your friend wants you to tune to: "Tune to WIOQ".
- Move eye to radio panel.
- Decide which of five radio buttons to depress.
- Move hand to the radio button.
- Depress the pushbutton.
- Say "Oh what!? This again?"

For Step 1, no measures of effectiveness were defined. However, the following objects were defined:

- CHANGE_STATION_STATUS. This object is associated with the CHANGE_STATION action. This object will be passed to each micromodel as the various behavioral steps are performed in the change station action. It has two characteristics: STEP (initialized to 1) and MICROMODEL_IN_USE (initialized to blank).

- RADIO. This is the object which stores the information relevant to this simulation concerning the radio. This object has two characteristics: CURRENT_STATION (initialized to WMMR) and DESIRED_STATION (initialized to WMMR).

The radio simulation does not require any actions to occur at every simulation time interval. Thus, Step 2 was skipped.

The radio simulation does require the action named CHANGE_STATION to occur at every simulation time interval that the current station is not the desired station. Thus for Step 3, the following rule was defined:

- The rule named CHANGE_STATION (rule #400) performs the change station action whenever the current station is not the desired station.

For Step 4, an event was defined to occur when the simulation time equalled one second. The event marked the time that the passenger requests a station change. The event invokes an action named REQUEST_STATION_CHANGE. This action, in turn, defines WIOQ as the desired radio station. (Note: When building the object named RADIO, the characteristic CURRENT_STATION is initialized to WMMR).

For Step 5, the simulation time unit was selected to be tenths of a second. The startup action named START_RADIO_SIMULATION starts the single operator rule (#400) required for the simulation.

Steps 6, 7, and 8 for creating, running, and viewing the results of the simulation can be accomplished by selecting the appropriate option on the HOS main screen. Figures 11-1 through 11-3 contain the RADIO simulation structure for simulation setup, event definition, and rule definition.

Simulation Setup

Simulation name:	radio
Simulation time interval:	tenths
Simulation description:	simulates changing a radio station from WMMR to WIOQ
Start action:	start_radio_simulation
Start simulation time:	00:00:00:00.0
Max simulation time:	00:00:00:10.0

Action: start_radio_simulation

Purpose: Starts operator rule 400.

Associated Objects: None.

Code:

```
COMMENT
          start_radio_simulation
ENDCOMMENT
     START OPERATOR 400
END
```

Figure 11-1. RADIO Simulation: Setup Simulation

Event	Description:	Tune to WIOQ
	Time:	00:00:00:01.0
	Do:	request_station_change

Action: request_station_change

Purpose: Establishes the condition to fire operator rule 400 by changing the value of the desired station to WIOQ. Note: The desired station of the radio is initialized to WMMR in the object editor.

Associated Objects:

Name	Characteristic	Initial Value
radio		
	desired_station	WMMR

Code:

```
COMMENT
        request_station_change
ENDCOMMENT
DEFINITIONS
        OBJECT                  radio
        CHARACTERISTIC          desired_station
ENDDEFINITIONS
PUT WIOQ IN desired_station OF radio
END
```

Figure 11-2. RADIO Simulation: Event Definition

Operator Rule 400 IF: desired_station OF radio NOT_EQUAL_TO current_station OF radio
DO: change_station
UNTIL: desired_station OF radio EQUALS current_station OF radio

Operator Action: change_station

Purpose: Contains the behavioral steps associated with tuning the radio to a new station.

Sequential Operator Steps:

Step	User Action	Micromodel/Action
1	Listen to the verbal request to change the radio station: "Tune to WIOQ."	HMM_listen
2	Move the eye to the radio panel.	HMM_eye_movement
3	Decide which of five radio buttons to depress.	HMM_decision
4	Move the hand to the appropriate button.	HMM_right_hand_ movement
5	Depress the radio pushbutton.	HMM_right_hand_ manip
6	Say "Oh what? This again?"	HMM_read_aloud
7	Set conditions to wrapup the change_ station action. Set the current_station OF radio to WIOQ so that rule 400 stops firing. Set the action step counter to one to prepare for the next time this action is called.	

Figure 11-3. RADIO Simulation: Rule Definition

Associated Objects:

Name	Characteristic	Initial Value
radio		
	current_station	WMMR
	desired_station	WMMR
change_station_status		
	step	1
	micromodel_in_use	blank

Code:

```
COMMENT
                                        change_station
    This action contains the behavioral steps associated with
    changing the radio station:
    1. listen to what station your friend wants you to tune to: "Tune to WIOQ."
    2. move eye to radio panel
    3. decide which of 5 radio buttons to press
    4. move hand to radio button
    5. depress button
    6. say "Oh what!? This again?
ENDCOMMENT
DEFINITIONS
    OBJECT              change_station_status, radio
    CHARACTERISTIC      step, current_station
    DECIMAL             distance, width
    WHOLE               number_of_words,
                        level_SN, level_noise_freq, alternatives
    ALPHABETIC          control_type, vocabulary_size
ENDDEFINITIONS
IF step OF change_station_status EQUALS 1 THEN
    SET number_of_words TO [6]
    SET level_SN TO [9]
    SET level_noise_freq TO [10]
    USING change_station_status, number_of_words, level_SN,
        level_noise_freq DO HMM_listen
ENDIF
```

Figure 11-3. RADIO Simulation: Rule Definition (continued)

```
IF step OF change_station_status EQUALS 2 THEN
        USING change_station_status DO HMM_eye_movement
ENDIF
IF step OF change_station_status EQUALS 3 THEN
        SET alternatives TO [5]
        USING change_station_status, alternatives DO HMM_decision
ENDIF
IF step OF change_station_status EQUALS 4 THEN
        SET distance to [12.0]
        SET width TO [0.5]
        USING change_station_status, distance, width
                DO HMM_right_hand_movement
ENDIF
IF step OF change_station_status EQUALS 5 THEN
        SET control_type TO [pushbutton]
        USING change_station_status, control_type
                DO HMM_right_hand_manip
ENDIF
IF step OF change_station_status EQUALS 6 THEN
        SET number_of_words to [4]
        SET vocabulary_size to [large]
        USING change_station_status, number_of_words, vocabulary_size
                DO HMM_read_aloud
ENDIF
IF step OF change_station_status EQUALS 7 THEN
        PUT WIOQ IN current_station OF radio
        PUT 1 IN step OF change_station_status
ENDIF
END
```

Figure 11-3. RADIO Simulation: Rule Definition (continued)

SAMPLER SIMULATION

SAMPLER simulates an operator monitoring two displays in parallel. The two monitoring tasks consist of the same human performance steps: (1) fixate on a reference point; (2) perceive the information contained in the reference, and (3) move the hand. One monitoring task may be stalemated by the other since an underlying behavioral resource is busy. For example, at simulation start, the higher priority monitoring task starts an eye movement while the lower priority task must wait until the eye is free. An emergency alarm is also simulated which will interrupt the monitoring tasks. Following the operator's response to that emergency, the two monitoring tasks will be reactivated.

For Step 1, three actions were defined. Two actions — SAM_MAIN_MONITOR and SAM_SECONDARY_MONITOR — simulated the normal conditions where the operator simultaneously performed two monitoring tasks. The third action — ALARM_RESPONSE — simulated the operator's task in responding to an emergency alarm. The normal monitoring actions contain the same behavioral steps:

- Move eye to a reference.
- Visually perceive the information displayed by the reference.
- Move hand.

The emergency response action contains the following steps:

- Move eye to a reference.
- Visually perceive the information displayed by the reference.
- Decide how to respond to the emergency.
- Move hand to the proper control.
- Depress a pushbutton.
- Return to normal monitoring.

For Step 1, no measures of effectiveness were defined. However, the following objects were defined:

- SAM_MAIN_MONITOR_STATUS. This object is associated with the SAM_MAIN_MONITOR action. This object will be passed to each HOS micromodel as the various behavioral steps are performed in the main monitor action. It has two characteristics: STEP (initialized to 1) and MICROMODEL_IN_USE (initialized to blank).
- SAM_SECONDARY_MONITOR_STATU. This object is associated with the SAM_SECONDARY_MONITOR action. This object will be passed to each HOS micromodel as the various behavioral steps are performed in the secondary monitor action. It has two characteristics: STEP (initialized to 1) and MICROMODEL_IN_USE (initialized to blank).
- SAM_PANIC_STATUS. This object is associated with the ALARM_RESPONSE action. This object will be passed to each HOS micromodel as the various behavioral steps are performed in the alarm response action. It has two characteristics: STEP (initialized to 1) and MICROMODEL_IN_USE (initialized to blank).
- SAM_ALARM This object stores the information relevant to this simulation concerning an emergency alarm. This object has one characteristic: STATUS (initialized to off).

The sampler simulation required that the two normal monitoring tasks occur simultaneously and continuously (barring any emergencies). Thus for Step 2, the following rules were defined:

- The rule named SAM_MAIN_MONITOR (rule #800) invokes the action

SAM_MAIN_MONITOR at every simulation time interval that this rule is active.

- The SAM_SECONDARY_MONITOR rule (rule #700) invokes the action SAM_SECONDARY_MONITOR at every simulation time interval that this rule is active.

For every simulation time interval that the above rules are active, each of the normal monitoring actions will be invoked.

The sampler simulation does require the action named ALARM_RESPONSE to occur at every simulation time interval that the alarm is on. Thus, for Step 3, the following rule was defined for the emergency condition:

- The rule named SAM_ALARM_MONITOR (rule #900) invokes the action ALARM_RESPONSE whenever it is active and the status of the alarm is on.

For Step 4, an event was defined to occur when the simulation time equalled two seconds. The event invokes an action named AUDIO_ALARM. This action, in turn, sets the status of the alarm to on and suspends the operator rules associated with the two normal monitoring tasks. Note that the AUDIO_ALARM action not only suspends operator rules #800 and #700, but also suspends all the HOS micromodels using the command:

```
DO HMM_micromodel_suspend
```

This was necessary so that all HOS micromodels which may be busy at that simulation time are set free for the emergency response action.

For Step 5, the simulation time unit was selected to be tenths of a second. The simulation startup action named SAM_START_SAMPLER starts all three operator rules (#900, #800, and #700) required for the simulation.

Steps 6, 7, and 8 for creating, running, and viewing the results of the simulation can be accomplished by selecting the appropriate option on the HOS main screen. Figures 11-4 through 11-8 contain the SAMPLER simulation structure for simulation setup, event definition, and rule definition.

Simulation Setup		
	Simulation name:	sampler
	Simulation time interval:	tenths
	Simulation description:	sampler simulation
	Start action:	SAM_start_sampler
	Start simulation time:	00:00:00:00.0
	Max simulation time:	00:00:00:10.0

Action: SAM_start_sampler

Purpose: Starts operator rules 700, 800, and 900.

Associated Objects: None.

Code:

```
COMMENT                          SAM_start_sampler
named within the SETUP SIMULATION EDITOR as the starting action
ENDCOMMENT
START OPERATOR 900
START OPERATOR 800
START OPERATOR 700
END
```

Figure 11-4. SAMPLER Simulation: Setup Simulation

Event	Description:	start the 'panic' alarm
	Time:	00:00:00:02.0
	Do:	audio_alarm

Action: audio_alarm

Purpose: Sets condition to fire operator rule 900. Suspends the rules associated with the operator's normal monitoring procedures (O800 and O700).

Associated Objects:

Name	Characteristic	Initial Value
SAM_alarm		
	status	off

Code:

```
COMMENT
                              audio_alarm
        This action starts the sequence required by the operator when
                        a "panic" alarm goes off
ENDCOMMENT
DEFINITIONS
        OBJECT                SAM_alarm
        CHARACTERISTIC        status
ENDDEFINITIONS
SUSPEND OPERATOR 800
SUSPEND OPERATOR 700
DO HMM_micromodel_suspend
PUT on IN status OF SAM_alarm
END
```

Figure 11-5. SAMPLER Simulation: Event Definition

Operator Rule 900 IF: status OF SAM_alarm EQUALS on
DO: alarm_response
UNTIL: status OF SAM_alarm EQUALS off

Operator Action: alarm_response

Purpose: Contains the behavioral steps associated with responding to an emergency alarm.

Sequential Operator Steps

Step	User Action	Micromodel/Action
1	Look at emergency display.	HMM_eye_movement
2	Perceive contents of the display.	HMM_visual_perceive
3	Make decision about appropriate response.	HMM_decision
4	Move right hand to pushbutton.	HMM_right_hand_ movement
5	Depress pushbutton.	HMM_right_hand_ manip
6	Set conditions to wrapup alarm_response (O900) and restart normal operator monitoring tasks (O800 and O700).	

Associated Objects:

Name	Characteristic	Initial Value
SAM_alarm		
	status	off
SAM_panic_status		
	step	1
	micromodel_in_use	blank

SAM_main_monitor_status (See Figure 11-7)
SAM_secondary_monitor_statu
(See Figure 11-8)

Figure 11-6. SAMPLER Simulation: Rule 900 Definition

Code:

```
COMMENT
                    alarm_response
            This action contains all the necessary steps for handling a
                    "panic" alarm situation.

        1.      move eye
        2.      determine what you are looking at
        3.      decide action based on this information
        4.      move hand to proper button
        5.      press a push button
        6.      return to previous tasks

ENDCOMMENT
DEFINITIONS
        OBJECT                  SAM_alarm,
                                SAM_panic_status,
                                SAM_main_monitor_status,
                                SAM_secondary_monitor_statu
        CHARACTERISTIC          status, step
        DECIMAL                 distance, width
        WHOLE                   alternatives
        ALPHABETIC              control_type
DEFINITIONS
IF step OF SAM_panic_status EQUALS 1 THEN
        USING SAM_panic_status DO HMM_eye_movement
ENDIF
IF step OF SAM_panic_status EQUALS 2 THEN
        USING SAM_panic_status DO HMM_visual_perceive
ENDIF
IF step OF SAM_panic_status EQUALS 3 THEN
        SET alternatives TO [3]
        USING SAM_panic_status, alternatives
                DO HMM_decision
ENDIF
```

Figure 11-6. SAMPLER Simulation: Rule 900 Definition (continued)

```
IF step OF SAM_panic_status EQUALS 4 THEN
        SET distance TO [20.0]
        SET width TO [1.0]
        USING SAM_panic_status, distance, width
                DO HMM_right_hand_movement
ENDIF
IF step OF SAM_panic_status EQUALS 5 THEN
        SET control_type TO [pushbutton]
        USING SAM_panic_status, control_type
        DO HMM_right_hand_manip
ENDIF
IF step OF SAM_panic_status EQUALS 6 THEN
        PUT off IN status OF SAM_alarm
        PUT 1 IN step OF SAM_panic_status
        SUSPEND OPERATOR 500
        START OPERATOR 800
        START OPERATOR 700
        PUT 1 IN step OF SAM_main_monitor_status
        PUT 1 IN step OF SAM_secondary_monitor_statu
ENDIF
END
```

Figure 11-6. SAMPLER Simulation: Rule 900 Definition (continued)

Operator Rule 800 IF: status OF HOS_run EQUALS on
DO: SAM_main_monitor
UNTIL: status OF HOS_run EQUALS off

Operator Action: SAM_main_monitor

Purpose: Contains the behavioral steps associated with monitoring a primary display.

Sequential Operator Steps

Step	User Action	Micromodel/Action
1	Look at display.	HMM_eye_movement
2	Perceive con[illegible] of the display.	HMM_visual_perceive
3	Make right hand movement.	HMM_right_hand_ movement
4	Put one in the step counter of this action so that the steps are repeated in the proper sequence.	

Associated Objects:

Name	Characteristic	Initial Value
SAM_main_monitor_status		
	step	1
	micromodel_in_use	blank

Figure 11-7. SAMPLER Simulation: Rule 800 Definition

Code:

```
COMMENT                                   SAM_main_monitor
               This action is called by the OPERATOR RULE 800 to simulate the
               operator performing substeps:
               1.    fixate on reference
               2.    visual perception of information contained in reference
               3.    hand movement
ENDCOMMENT
DEFINITIONS
       OBJECT                     SAM_main_monitor_status
       CHARACTERISTIC             step
       DECIMAL                    distance, width
ENDDEFINITIONS
IF step OF SAM_main_monitor_status EQUALS 1 THEN
       USING SAM_main_monitor_status DO HMM_eye_movement
ENDIF
IF step OF SAM_main_monitor_status EQUALS 2 THEN
       USING SAM_main_monitor_status DO HMM_visual_perceive
ENDIF
IF step OF SAM_main_monitor_status EQUALS 3 THEN
       SET distance TO [10.]
       SET width TO [0.5]
       USING SAM_main_monitor_status, distance, width
              DO HMM_right_hand_movement
ENDIF
IF step OF SAM_main_monitor_status EQUALS 4 THEN
       PUT 1 IN step OF SAM_main_monitor_status
ENDIF
END
```

Figure 11-7. SAMPLER Simulation: Rule 800 Definition (continued)

Operator Rule 700 IF: status OF HOS_run EQUALS on
DO: SAM_secondary_monitor
UNTIL: status OF HOS_run EQUALS off

Operator Action: SAM_secondary_monitor

Purpose: Contains the behavioral steps associated with monitoring a secondary display.

Sequential Operator Steps

Step	User Action	Micromodel/Action
1	Look at display.	HMM_eye_movement
2	Perceive contents of the display.	HMM_visual_perceive
3	Make right hand movement.	HMM_right_hand_movement
4	Put one in the step counter of this action so that the steps are repeated in the proper sequence.	

Associated Objects:

Name	Characteristic	Initial Value
SAM_secondary_monitor_statu		
	step	1
	micromodel_in_use	blank

Figure 11-8. SAMPLER Simulation: Rule 700 Definition

Code:

```
COMMENT                              SAM_secondary_monitor
                 This action is called by the OPERATOR RULE 700 to simulate the
                 operator performing substeps:
                 1.    fixate on reference
                 2.    visual perception of information contained in reference
                 3.    hand movement
ENDCOMMENT
DEFINITIONS
        OBJECT                   SAM_secondary_monitor_statu
        CHARACTERISTIC           step
        DECIMAL                  distance, width
ENDDEFINITIONS
IF step OF SAM_secondary_monitor_statu EQUALS 1 THEN
        USING SAM_secondary_monitor_statu DO HMM_eye_movement
ENDIF
IF step OF SAM_secondary_monitor_statu EQUALS 2 THEN
        USING SAM_secondary_monitor_statu DO HMM_visual_perceive
ENDIF
IF step OF SAM_secondary_monitor_statu EQUALS 3 THEN
        SET distance TO [10.]
        SET width TO [0.5]
        USING SAM_secondary_monitor_statu, distance, width
                DO HMM_right_hand_movement
ENDIF
IF step OF SAM_secondary_monitor_statu EQUALS 4 THEN
        PUT 1 IN step OF SAM_secondary_monitor_statu
ENDIF
END
```

Figure 11-8. SAMPLER Simulation: Rule 700 Definition (continued)

A. GENERAL INSTALLATION INSTRUCTIONS

This section describes the utilities provided on the HOS utility diskette and provides general information on how to install the mouse, RAMdisk, and Microsoft C compiler on your computer. For exact details on the procedures to install these items, consult the manual that accompanied the product. These guidelines assume that the installation is on a hard disk, Drive C.

Included on the HOS IV Utilities Disk are the following files:

```
INSTALL.BAT
SETUPSYS.EXE
BUILDHOS.BAT
MOVE.BAT
REST_HOS.BAT
BACK_HOS.BAT
AUTOEXEC.BAT
CONFIG.SYS
MOUSEX.COM
MSMOUSE.COM
BACKUP.COM
RESTORE.COM
```

Note: some of these files are included on this diskette as a precaution in case the file is accidentally erased from the HOS IV directories. The function of each of these files is described below.

INSTALL.BAT

This file is used to initally install the HOS IV System. To use this file, be sure that the HOS IV Utilities disk is in the A: drive then type:

```
A:install
```

and press the Return/Enter key. The user will be prompted to insert each Backup Diskette into drive A:. (The backup diskettes are those that contain the actual HOS IV System and are marked "HOS IV

BACKUP 1 of n" through "HOS IV BACKUP n of n"). Note: This file should only be used once — at the INITIAL installation. If it is executed more than once, the HOS IV software will not be damaged but any data produced by the user using HOS IV will be destroyed.

SETUPSYS.EXE

This file is used to initialize the newly installed HOS IV system. Once HOS IV has been installed, this file can be used from the HOSIV directory. To execute this file, type:

```
cd \HOSIV
```

to change directories to the HOS IV system. Then type:

```
setupsys
```

to execute the file. Several questions are displayed to the user, each of which requires a single letter response Y for yes or N for no. It is important that the questions are answered in order for HOS IV to function properly. If the user wants to change the response to any of these questions, the setupsys program can be executed again and the correct responses entered. Note:This initialization assumes that the user has installed HOS IV on a hard disk referenced as drive C: If this is not the case, change the directory to the one containing HOS IV by entering:

```
cd <drive letter:>\HOSIV
```

where <drive letter> is replaced with the drive designation for the device. (<drive letter> should be E: for a Bernoulli Box).

BUILDHOS.BAT

This file is used to recreate the HOS IV system from the C language source code. To use, change directories to \HOSIV (see setupsys for more information about changing directories) and then type:

```
cd source
```

and press the Return/Enter key. Then type:

```
buildhos
```

and press the Return/Enter key. This file requires about**4hours** to complete but once started, no responses are required from the user so the process can be left unattended. Note: This utility creates a multitude of temporary files during the code compilation. Many of these temporary files are quite large, so there must be **at least 3 Megabytes of free** space available on the device where the source code is located. The temporary files are removed as the final action of this utility.

MOVE.BAT

This file is used to complement BUILDHOS.BAT by copying the files created from the source code to the \HOSIV directory so that they may be used. To use MOVE.BAT, change directories to the directory in which the source code is located (see BUILDHOS.BAT for more information). Then type:

```
move
```

and press the Return/Enter key. The display will then print "1 File(s) Copied" several times, and the system prompt will return.

BACK_HOS.BAT

This file is used to make a backup of the HOS IV system including any data created by the user. To use, change directories to the \HOSIV directory, then type:

```
back_hos
```

and press the Return/Enter key. The computer will prompt the user to insert backup diskette 01 in the A: drive. As each diskette is filled with files, the computer will prompt the user to insert the next diskette. Note: BACK_HOS.BAT destroys all information on the diskette used. Therefore, it is important that the diskettes used for the backup procedure not contain any information the user still needs. Also, once a backup diskette is created by BACK_HOS, it must not be written to by the user (such as by copying a file onto it) as this destroys the backup information. (It is, however, acceptable to reuse backup diskettes as long as the data previously stored on the diskette is no longer needed.)

REST_HOS.BAT

This file is used to restore the HOS IV system from a series of diskettes created by the BACK_HOS utility. To use, change directories to the \HOSIV directory and then type:

```
rest_hos
```

and press the Return/Enter key. The computer will prompt the user to insert each restore diskette in the A: drive. As each diskette is read, the computer will prompt the user to insert the next. Note: If the system uses a Bernoulli Box for HOS IV, the REST_HOS will also prompt the user to insert restore diskette 01 in E:. If this question is asked, simply press the Return/Enter key.

AUTOEXEC.BAT

This file is included as an example of the commands contained in a typical autoexec.bat file.

CONFIG.SYS

This file is included as an example of the commands typically included in a config.sys file. This example assumes a Bernoulli Box is being used. The file may be modified as necessary but the statements that begin "device=" must appear in the same order as in this file and they must be the first lines in the file. Any additional devices may be installed after these declarations. Note: If your system does not use a Bernoulli Box, the line that reads "device=IDRIVE.sys" may be removed from this file. If your system uses a different Ramdrive device (this config.sys assumes it is called "ramdrive"), follow the instructions enclosed with your ramdrive for installation and then be sure that the references to the Ramdrive that were put into the config.sys come BEFORE references to any other device. For example, if your Ramdrive added the lines:

```
DEVICE = EMM.SYS AT D000 208
DEVICE = QUIKMEM2.sys 1024
```

to your config.sys, it is imperative that these lines appear above the line that says "device = IDRIVE.SYS". Otherwise, the HOS IV system cannot function properly.

MOUSEX.COM

This file is already installed in the \HOSIV directory. It is used to control the mouse features of the HOS IV system.

MSMOUSE.COM

This file may not be required by all users. If you are using a MicroSoft Mouse (or a mouse from Mouse Systems) this file should be copied to the \HOSIV directory.

BACKUP.COM AND RESTORE.COM

These files are included for compatability. If the INSTALL utility on your computer cannot read the HOS IV diskettes, copy both of these files into the \HOSIV directory and copy INSTALL.BAT to \HOSIV. Then change directories to \HOSIV and execute install again. If the diskettes still cannot be read, the diskettes may be damaged. Consult a DOS expert to determine what can be done.

MOUSE INSTALLATION

The following outlines the procedure to install the mouse driver:

- Place the mouse driver disk that accompanied the mouse in the A diskette drive.
- Enter DIR A:*.SYS to obtain a list of all the files on the mouse driver disk.
- The directory should contain a file named something like MOUSE.SYS.
- Enter COPY A:MOUSE.SYS C:\ to copy the MOUSE.SYS file to the root node.
- Using a text editor, create or modify the CONFIG.SYS file (at the root node) to insert a line as follows:

  ```
  DEVICE = MOUSE.SYS
  ```
- Reboot the computer by depressing CTRL, ALT, DELETE simultaneously.
- Enter the following: "MOUSEX+".

RAMDISK INSTALLATION

The RAMdisk is used by HOS to emulate a disk drive with the computer's random access memory. HOS requires at least a full megabyte of RAMdisk in addition to 640Kb for DOS. The documentation that accompanied the expanded memory board should be consulted to determine how to install the RAMdisk. HOS requires that the RAMdisk be installed as drive D.

MICROSOFT C INSTALLATION

The instructions in the Microsoft C manual entitled 'Hard-Disk Setup Procedure' should be followed to install Microsoft C. HOS requires that the C compiler be in the NEWC subdirectory. Therefore the instructions contained in the Microsoft C manual for building the subdirectories should be modified to be as follows (the additional commands are shown in bold typeface):

CD \

MKDIR NEWC

CD \NEWC

MD BIN

MD LIB

MD INCLUDE

MD INCLUDE\SYS

Microsoft C contains three sets of library files — small, medium and large. HOS requires that the large model library be installed.

B. DEFAULT MICROMODEL PARAMETERS

This section contains a list of the objects associated with HOS micromodels and their characteristics which store timing and error parameters as shown in Figure B-1. Default values are also listed.

MICROMODEL NAME	OBJECT NAME	CHARACTERISTIC NAME	DEFAULT
HMM_decision	HMM_decision_status	decision_constant	0.15 seconds
HMM_eye_movement	HMM_eye_movement_status	time	0.17 seconds
HMM_fatigue	HMM_fatigue_status	update_interval	300 seconds
		hours_per_update_interval	0.083 hours
		cumulative_hours_fatigue	0 hours
HMM_right_hand_movement	HMM_right_hand_move_status	move_constant	0.10 seconds
HMM_left_hand_movement	HMM_left_hand_move_status	move_constant	0.10 seconds
HMM_handprint	HMM_handprint_status	time_per_character	0.75 seconds
HMM_listen	HMM_listen_status	words_per_second	2.4 seconds
	HMM_listen_error_data (SET of 12)		
	HMM_listen_error_data (item 1)	signal_to_noise_level	9 decibels
		noise_interrupt_frequency	1 HZ
		error_rate	0.10
	HMM_listen_error_data (item 2)	signal_to_noise_level	9 decibels
		noise_interrupt_frequency	10 HZ
		error_rate	0.07
	HMM_listen_error_data (item 3)	signal_to_noise_level	9 decibels
		noisa_interrupt_frequency	100 HZ
		error_rate	0.10
	HMM_listen_error_data (item 4)	signal_to_noise_level	0 decibels
		noise_interrupt_frequency	1 HZ
		error_rate	0.20
	HMM_listen_error_data (item 5)	signal_to_noise_level	0 decibels
		noise_interrupt_frequency	10 HZ
		error_rate	0.12
	HMM_listen_error_data (item 6)	signal_to_noise_level	0 decibels
		noise_interrupt_frequency	100 HZ
		error_rate	0.25
	HMM_listen_error_data (item 7)	signal_to_noise_level	-9 decibels
		noise_interrupt_frequency	1 HZ
		error_rate	0.35
	HMM_listen_error_data (item 8)	signal_to_noise_level	-9 decibels
		noise_interrupt_frequency	10 HZ

MICROMODEL NAME	OBJECT NAME	CHARACTERISTIC NAME	DEFAULT
	HMM_listen_error_data (item 9)	error_rate	0.25
		signal_to_noise_level	-9 decibels
		noise_interrupt_frequency	100 HZ
	HMM_listen_error_data (item 10)	error_rate	0.65
		signal_to_noise_level	-18 decibels
		noise_interrupt_frequency	1 HZ
	HMM_listen_error_data (item 11)	error_rate	0.42
		signal_to_noise_level	-18 decibels
		noise_interrupt_frequency	10 HZ
	HMM_listen_error_data (item 12)	error_rate	0.28
		signal_to_noise_level	-18 decibels
		noise_interrupt_frequency	100 HZ
		error_rate	0.96
HMM_right_hand_manip	HMM_control_data (SET of 4)		
	HMM_control_data (item 1)	control_type	pushbutton
		time	0.40 seconds
	HMM_control_data (item 2)	control_type	toggle
		time	0.40 seconds
	HMM_control_data (item 3)	control_type	rotary_dial
		time	0 73 seconds
	HMM_control_data (item 4)	control_type	trackball
		time	0.10 seconds
HMM_left_hand_manip	HMM_control_data (SET of 4)		
	HMM_control_data (item 1)	control_type	pushbutton
		time	0.40 seconds
	HMM_control_data (item 2)	control_type	toggle
		time	0.40 seconds
	HMM_control_data (item 3)	control_type	rotary_dial
		time	0.73 seconds
	HMM_control_data (item 4)	control_type	trackball
		time	0.10 seconds
HMM_memory_store	HMM_memory_data	storage_time	0.10 seconds
		retr_label_time	0.07 seconds

MICROMODEL NAME	OBJECT NAME	CHARACTERISTIC NAME	DEFAULT
		search_rate_per_item	0.047 seconds
HMM_memory_retrieve	HMM_memory_data		
HMM_read_aloud	HMM_vocabulary_data (SET of 2)		
	HMM_vocabulary_data (item 1)	size_of_vocab	small
		words_per_second	3.4 seconds
	HMM_vocabulary_data (item 2)	size_of_vocab	large
		words_per_second	2.4 seconds
HMM_visual_perceive	HMM_visual_perceive_status	time	0.34 seconds
HMM_walking	HMM_walking_status	time_per_foot	0.19 seconds

C. TROUBLESHOOTING

This section describes some of the error conditions that may occur while building or executing a HOS simulation. The errors and the appropriate corrective actions are organized in this appendix by the simulation activity during which the error may occur.

ENTERING HOS

To enter the HOS-IV simulation system, the user types the *runhos* command:

```
C:\HOSIV>runhos
```

Upon receipt of this command, a batch file is executed which activates a mouse driver and also deletes two files associated with the last HOS session, specifically the files *out.out* and *hoserror.err.* The operating system will momentary flash these messages while entering HOS:

```
C:\HOSIV>mousex +
Microsoft mouse driver extension
Configured for +

C:\HOSIV>del hoserror.err
File not found

C:\HOSIV>hosiv > out.out

C:\HOSIV>type hoserror.err | more
File not found
```

This is a normal condition.

USING HOS EDITORS

While exiting and entering the various HOS Editors and simulation options (e.g., Object Editor, Action Editor, Create Simulation, Run Simulation), the HOS-IV system may abnormally terminate. If this occurs, reenter HOS using the *runhos* command.

```
C:\HOSIV>runhos
```

No files are lost as a result of the abnormal termination.

USING THE CUT AND PASTE OPTIONS OF THE ACTION EDITOR

Occasionally, while using the Action Editor's cut and paste options (of the pull down menu named EDIT), it may appear that the text is inserted incorrectly. If this occurs, depress the END or HOME keys on the numeric keypad and then move the cursor to the original location of the text insertion. The text should then be correctly displayed. If not, manually correct the text without use of the cut or paste options.

TRANSLATING ACITONS IN THE ACTION EDITOR

The Action Editor's translate option (of the pull down menu named FILE) executes two steps:

1. Translate the HOS Action Language (of the ACTION file currently open) into a module written in the C language.
2. Compile the C language module into IBM machine language instructions and insert the machine language module into the HOS action library (ACTION.LIB).

The first step is executed while the screen displays the blue window which reads:

HOS Action Language Translating.

If the following message is displayed:

> The translation was not successful,

look at the file named TRANSLATOR OUTPUT using the View File option of the pull down menu named User Aids. This file lists the HAL translation error — usually a syntax error, a typing error, or an undefined variable error.

The second step, translating C into machine code, is executed when the screen becomes black and lists a line defining the version of the Microsoft compiler. If the following message is displayed upon a return to the HOS Action Editor screen:

> There has been a serious error with the
> HPL translation. Contact a HOS expert.

then exit the ACTION Editor and exit the HOS main screen. In the operating system, list the file named *out.out* with the command:

```
C:\HOSIV>type out.out.
```

If the message in *out.out* reads "OUT OF HEAP SPACE", reenter HOS and retranslate the action. If the same error occurs, then the action contains too many lines. Break the action into two modules or eliminate unnecessary statements or variables. (The Microsoft C Compiler limits the lines of a single module to less than 150 lines. There is not a direct correlation between the number of lines in a HOS action and the number of lines in the associated C language module. Typically, the C language file contains more lines than the HAL action file.) It is recommended that HOS action files have a maximum of 100 lines.

Alternatively, *out.out* may list C compiler error messages concerning variable definition such as:

```
\hosiv\hosc\p0000120.c(65) : error 65,
    'emitter_store' : undefined
\hosiv\hosc\p0000120.c(65) : warning 49,
    'parameter' : indirection to different types
```

In this case, reenter HOS and the Action Editor. Open the file named SYMBOL TABLE. If object names, object set names, and variable names are not correctly defined by type (for example, an object is incorrectly defined as a local decimal variable), then the names of the objects and variables used in the action may be too similar (for example, 'emitter_store' and 'emitter_index'. The HOS action translator only hashes variable names across the first eight characters. Rename the variables so that the first eight characters are unique (for example, 'emit_store' and 'emit_index') and retranslate.

RUNNING A SIMULATION

If while running a simulation, the simulation clock stops and the action, rule, and event windows remain fixed for several minutes, then an endless loop may be occurring. If this happens, the simulation can only be terminated by simultaneously depressing the Shift, Alt, and Delete keys. Investigate the logic of the action which was last displayed on the run simulation screen. Typically, the action contains a WHILE/ENDWHILE statement group without a mechanism for exiting the WHILE loop, that is, the exiting condition is never met. Note: The user is cautioned when the terminating condition of a WHILE loop is based on the comparison of two decimal numbers. The floating point representations of two decimal values may appear to be equal in a PRINT statement. However, if even the least significant bits of the computer's internal representation of the two numbers differ, then the two variables will not be considered as equal.

Another example of an endless loop situation can occur within an operator action when a series of HOS micromodels are sequentially called. Recall that when a HOS micromodel is accessed, the user sends the name of the object associated with the calling action (for example, the *change_station_status* object associated with the *change_station* action.) When the micromodel is finished executing, the step characteristic of the calling action object is incremented by one. If while building an operator action, the user accidentally skips a step number, then the action will not execute the final steps. For example,

```
IF step OF change_status_status EQUALS 1 THEN
    USING change_station_status DO
        HMM_eye_movement
ENDIF
IF step OF change_station_status EQUALS 3 THEN
    USING change_station_status DO
        HMM_visual_perceive
ENDIF
```

In this case, the eye movement micromodel updates the step number to two when it is completed. However, since a case defining what is to occur at step number two does not exist, subsequent steps will never be executed as a mechanism to update the step number to three does not exist.

Another example of an endless loop is illustrated below for the operator action named report_contact:

```
IF step OF report_contact_status EQUALS 1 THEN
   IF number OF contacts GREATER_THAN 0 THEN
       USING report_contact_status DO send_data
   ENDIF
ENDIF
IF step OF report_contact_status EQUALS 2 THEN
   .
   .
ENDIF
```

In this case, if the number of contacts equals zero, send_data will not be executed and the report_contact action will remain in step 1 (assuming that the send_data action, upon its completion, is the only mechanism for incrementing the step of report_contact_status). To account for the case when the number of contacts equals zero, the code could be rewritten as follows:

```
IF step OF report_contact_status EQUALS 1 THEN
   IF number OF contacts GREATER_THAN 0 THEN
        USING report_contact_status DO send_data
   ENDIF
   ELSE
        PUT 2 IN step OF report_contact_status
   ENDELSE
ENDIF
IF step OF report_contact_status EQUALS 2 THEN
   .
   .
ENDIF
```

USING VIEWRESULTS

Following selection of the ViewResults option on the main HOS menu screen, a blank ViewResults screen will appear. Control of the mouse cursor will not be available immediately while the HOS-IV system builds the standard set of output analysis files. The length of time required for HOS to build these files depends on the length of the simulation. The output files of small simulations (100 time units) are usually built within 20-30 seconds. Long simulations may require several minutes to build the ViewResults files.